

175-TP-511-002

HDF-EOS Interface Based on HDF5, Volume 2: Function Reference Guide

Technical Paper

July 2001

Prepared Under Contract NAS5-60000

RESPONSIBLE ENGINEER

David Wynne /s/ 7/20/01
David Wynne, Alex Muslimov, Date
Abe Taaheri, Ray Milburn
EOSDIS Core System Project

SUBMITTED BY

Larry Klein /s/ 7/20/01
Darryl Arrington, Manager Toolkit Date
Larry Klein, Manager Toolkit
EOSDIS Core System Project

Raytheon Company
Upper Marlboro, Maryland

This page intentionally left blank.

Preface

This document is a Users Guide for HDF-EOS (Hierarchical Data Format - Earth Observing System) library tools. The version described in this document is HDF-EOS Version 5.1. The software is based on HDF5, a new version of HDF provided by NCSA. HDF5 is a complete rewrite of the earlier HDF4 version, containing a different data model and user interface. HDF-EOS V5.1 incorporates HDF5, and keeps the familiar HDF4-based interface. There are a few exceptions and these exceptions are described in this document.

HDF is the scientific data format standard selected by NASA as the baseline standard for EOS. This Users Guide accompanies Version 5.1 software, which is available to the user community on the EDHS1 server. This library is aimed at EOS data producers and consumers, who will develop their data into increasingly higher order products. These products range from calibrated Level 1 to Level 4 model data. The primary use of the HDF-EOS library will be to create structures for associating geolocation data with their associated science data. This association is specified by producers through use of the supplied library. Most EOS data products which have been identified, fall into categories of point, grid or swath structures, the latter two of which are implemented in the current version of the library. Services based on geolocation information will be built on HDF-EOS structures. Producers of products not covered by these structures, e.g. non-geolocated data, can use the standard HDF libraries.

In the ECS (EOS Core System) production system, the HDF-EOS library will be used in conjunction with SDP (Science Data Processing) Toolkit software. The primary tools used in conjunction with HDF-EOS library will be those for metadata handling, process control and status message handling. Metadata tools will be used to write ECS inventory and granule specific metadata into HDF-EOS files, while the process control tools will be used to access physical file handles used by the HDF tools. (SDP Toolkit Users Guide for the ECS Project, November 2000, 333-CD-600-001).

HDF-EOS is an extension of NCSA (National Center for Supercomputing Applications) HDF5 and uses HDF5 library calls as an underlying basis. Version 5-1.4.1 of HDF5 is used. The library tools are written in the C language and a FORTRAN interface is provided. The current version contains software for creating, accessing and manipulating Grid, Point and Swath structures. This document includes overviews of the interfaces, and code examples. HE5View, the HDF-EOS viewing tool, has been revised to accommodate the current version of the library.

Note that HDF-EOS V2.X, based on HDF4 is also available. HDF-EOS 2.X is a separate library. Both versions of HDF-EOS will be supported by ECS.

Technical Points of Contact within EOS are:

Larry Klein, larry@eos.hitc.com

David Wynne, davidw@eos.hitc.com

Alex Muslimov, amuslimo@eos.hitc.com

An email address has been provided for user help:

pgstlkit@eos.hitc.com

Any questions should be addressed to:

Data Management Office

The ECS Project Office

Raytheon Systems Company

1616 McCormick Drive

Upper Marlboro, MD 20774-5301

Abstract

This document will serve as the user's guide to the HDF-EOS file access library based on HDF5. HDF refers to the scientific data format standard selected by NASA as the baseline standard for EOS, and HDF-EOS refers to EOS conventions for using HDF. This document will provide information on the use of the three interfaces included in HDF-EOS – Point, Swath, and Grid – including overviews of the interfaces, and code examples. This document should be suitable for use by data producers and data users alike.

Keywords: HDF-EOS, HDF5, Metadata, Standard Data Format, Standard Data Product, Disk Format, Grid, Point, Swath, Projection, Array, Browse

This page intentionally left blank

Contents

Preface

Abstract

1. Introduction

| | | |
|-------|----------------------------------|------|
| 1.1 | Purpose..... | 1-1 |
| 1.2 | Organization..... | 1-1 |
| 1.3 | Point Data..... | 1-1 |
| 1.3.1 | The Point Data Interface | 1-1 |
| 1.3.2 | List of PT API Routines..... | 1-2 |
| 1.4 | Swath Data..... | 1-3 |
| 1.4.1 | The Swath Data Interface..... | 1-3 |
| 1.4.2 | List of SW API Routines | 1-3 |
| 1.5 | Grid Data..... | 1-6 |
| 1.5.1 | The Grid Data Interface | 1-6 |
| 1.5.2 | List of Grid API Routines | 1-6 |
| 1.6 | GCTP Usage | 1-8 |
| 1.6.1 | GCTP Projection Codes..... | 1-8 |
| 1.6.2 | UTM Zone Codes | 1-10 |
| 1.6.3 | GCTP Spheroid Codes..... | 1-10 |
| 1.6.4 | GCTP Projection Parameters | 1-11 |

2. Function Reference

| | | |
|-------|---------------------------------|------|
| 2.1 | Format | 2-1 |
| 2.1.1 | Point Interface Functions | 2-1 |
| 2.1.2 | Swath Interface Functions..... | 2-45 |

| | | |
|-------|--------------------------------|-------|
| 2.1.3 | Grid Interface Functions | 2-148 |
| 2.1.4 | HDF-EOS Utility Routines | 2-226 |

List of Table

| | | |
|------|--|------|
| 1-1. | Summary of the Point Interface | 1-2 |
| 1-2. | Summary of the Swath Interface..... | 1-4 |
| 1-3. | Summary of the Grid Interface | 1-6 |
| 1-4. | Projection Transformation Package Projection Parameters..... | 1-11 |

Abbreviations and Acronyms

1. Introduction

1.1 Purpose

The *HDF-EOS Software Reference Guide for the ECS Project* was prepared under the Earth Observing System Data and Information System (EOSDIS) Core System (ECS), Contract (NAS5-60000).

This software reference guide is intended for use by anyone who wishes to use the HDF-EOS library to create or read EOS data products. Users of this document will include EOS instrument team science software developers and data product designers, DAAC personnel, and end users of EOS data products such as scientists and researchers.

1.2 Organization

This paper is organized as follows:

- Section 1 Introduction - Presents Scope and Purpose of this document
- Section 2 Function Reference
- Abbreviations and Acronyms

1.3 Point Data

The PT interface consists of routines for storing, retrieving, and manipulating data in point data sets.

1.3.1 The Point Data Interface

All C routine names in the point data interface have the prefix “HE5_PT” and the equivalent FORTRAN routine names are prefixed by “he5_pt.” The PT routines are classified into the following categories:

- *Access routines* initialize and terminate access to the PT interface and point data sets (including opening and closing files).
- *Definition routines* allow the user to set key features of a point data set.
- *Basic I/O routines* read and write data and metadata to a point data set.
- *Index I/O routines* read and write information which links two tables in a point data set.
- *Inquiry routines* return information about data contained in a point data set.
- *Subset routines* allow reading of data from a specified geographic region.

1.3.2 List of PT API Routines

The PT function calls are listed in Table 1-1 and are described in detail in the Software Reference Guide that accompanies this document. The page number column in the following table refers to the Software Reference Guide.

Table 1-1. Summary of the Point Interface (1 of 2)

| Category | Routine Name | | Description | Page Nos. |
|------------|---------------------|--------------------|---|-----------|
| | C | FORTRAN | | |
| Access | HE5_PTopen | he5_ptopen | Creates a new file or opens an existing one | 2-31 |
| | HE5_PTcreate | he5_ptcreate | Creates a new point data set and returns a handle | 2-06 |
| | HE5_PTattach | he5_ptattach | Attaches to an existing point data set | 2-02 |
| | HE5_PTd detach | he5_ptdetach | Releases a point data set and frees memory | 2-15 |
| | HE5_PTclose | he5_ptclose | Closes the HDF-EOS file and deactivates the point interface | 2-05 |
| Definition | HE5_PTdeflevel | he5_ptdeflevel | Defines a level within the point data set | 2-07 |
| | HE5_PTdeflinkage | he5_ptdeflinkage | Defines link field to use between two levels | 2-14 |
| | HE5_PTwritelev el | he5_ptwritelev el | Writes (appends) full records to a level | 2-44 |
| | HE5_PTreadlev el | he5_ptreadlev el | Reads data from the specified fields and records of a level | 2-35 |
| Basic I/O | HE5_PTupdatelev el | he5_ptupdatelev el | Updates the specified fields and records of a level | 2-39 |
| | HE5_PTwriteattr | he5_ptwriteattr | Creates or updates an attribute of the point data set | 2-40 |
| | HE5_PTwritegrpa ttr | he5_ptwritegrpatr | Writes/updates group attribute in a point | 2-42 |
| | HE5_PTwriteloca ttr | he5_ptwritelocattr | Write/updates local attribute in a point | 2-45 |
| | HE5_PTreadattr | he5_ptreadattr | Reads existing attribute of point data set | 2-32 |
| | HE5_PTreadgrpa ttr | he5_ptreadgrpatr | Reads group attribute from a point | 2-33 |
| | HE5_PTreadloca ttr | he5_ptreadlocattr | Reads local attribute from a point | 2-34 |
| | HE5_PTnlevels | he5_ptnlevels | Returns the number of levels in a point data set | 2-29 |
| | HE5_PTnrecs | he5_ptnrecs | Returns the number of records in a level | 2-30 |
| | HE5_PTnfields | he5_ptnfields | Returns number of fields defined in a level | 2-28 |
| | HE5_PTlevelinfo | he5_ptlevelinfo | Returns information about a given level | 2-26 |
| | HE5_PTlevelindx | he5_ptlevelindx | Returns index number for a named level | 2-25 |
| Inquiry | HE5_PTbcklinkinfo | he5_ptbcklinkinfo | Returns link field to previous level | 2-04 |
| | HE5_PTfwdlinkinfo | he5_ptfwdlinkinfo | Returns link field to following level | 2-16 |

Table 1-1. Summary of the Point Interface (2 of 2)

| | | | | |
|--|--------------------|--------------------|---|------|
| | HE5_PTgetlevelname | he5_ptgetlevelname | Returns level name given level number | 2-17 |
| | HE5_PTgetrecnums | None | Retrieves number of records in one level corresponding to a group of records in a different level | 2-18 |
| | HE5_PTattrinfo | he5_ptattrinfo | Returns information about point attributes | 2-03 |
| | HE5_PTgrpattrinfo | he5_ptgrpattrinfo | Returns information about point group attributes | 2-19 |
| | HE5_PTlocattrinfo | he5_ptlocattrinfo | Returns information about point local attributes | 2-27 |
| | HE5_PTinqattrs | he5_ptinqattrs | Retrieves number and names of point attributes | 2-20 |
| | HE5_PTinqgrptrs | he5_ptinqgrptrs | Retrieves number and names of group attributes | 2-21 |
| | HE5_PTinqlocatts | he5_ptinqlocatts | Retrieves number and names of local attributes defined | 2-22 |
| | HE5_PTinqpoint | he5_ptinqpoint | Retrieves number and names of points in file | 2-24 |
| | HE5_PTinqdatatype | he5_ptinqdatatype | Returns data type information about specified level in point | 2-37 |

1.4 Swath Data

The SW (*Swath*) interface consists of routines for storing, retrieving, and manipulating data in swath data sets. This interface is tailored to support time-ordered data such as satellite swaths (which consist of a time-ordered series of scanlines), or profilers (which consist of a time-ordered series of profiles). See the Users' Guide, Volume 1 that accompanies this document for more information.

1.4.1 The Swath Data Interface

All C routine names in the swath data interface have the prefix “HE5_SW” and the equivalent FORTRAN routine names are prefixed by “he5_sw.” The SW routines are classified into the following categories:

- **Access routines** initialize and terminate access to the SW interface and swath data sets (including opening and closing files).
- **Definition** routines allow the user to set key features of a swath data set.
- **Basic I/O** routines read and write data and metadata to a swath data set.
- **Inquiry** routines return information about data contained in a swath data set.
- **Subset** routines allow reading of data from a specified geographic region.

1.4.2 List of SW API Routines

The SW function calls are listed below in Table 1-2 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.

Table 1-2. Summary of the Swath Interface (1 of 3)

| Category | Routine Name | | Description | Page Nos. |
|-------------|---------------------|-----------------|---|-----------|
| | C | FORTRAN | | |
| Access | HE5_SWopen | he5_swopen | Opens or creates HDF file in order to create, read, or write a swath | 2-113 |
| | HE5_SWcreate | he5_swcreate | Creates a swath within the file | 2-54 |
| | HE5_SWattach | he5_swattach | Attaches to an existing swath within the file | 2-50 |
| | HE5_SWdetach | he5_swdetach | Detaches from swath interface | 2-77 |
| | HE5_SWclose | he5_swclose | Closes file | 2-52 |
| Definition | HE5_SWdefdim | he5_swdefdim | Defines a new dimension within the swath | 2-64 |
| | HE5_SWdefdimmap | he5_swdefmap | Defines the mapping between the geolocation and data dimensions | 2-66 |
| | HE5_SWdefidxmap | he5_swdefimap | Defines a non-regular mapping between the geolocation and data dimension | 2-70 |
| | HE5_SWdefgeofield | he5_swdefgfld | Defines a new geolocation field within the swath | 2-68 |
| | HE5_SWdefdatafield | he5_swdefdfld | Defines a new data field within the swath | 2-62 |
| | HE5_SWdefcomp | he5_swdefcomp | Defines a field compression scheme | 2-60 |
| | HE5_SWdefchunk | he5_swdefchunk | Defines chunking parameters | 2-57 |
| | HE5_SWwritegeometa | he5_swwrgmeta | Writes field metadata for an existing swath geolocation field | 2-139 |
| | HE5_SWwritedatameta | he5_swwrdmeta | Writes field metadata for an existing swath data field | 2-134 |
| | HE5_SWdefcomchunk | he5_swdefcomch | Defines compression with automatic chunking | 2-58 |
| | HE5_SWsetalias | he5_swsetalias | Defines alias for data field | 2-126 |
| | HE5_SWdropalias | he5_swdrpalias | Removes alias from the list of field aliases | 2-79 |
| | HE5_SWfldrename | he5_swfldrnm | Changes the field name | 2-88 |
| Basic I/O | HE5_SWaliasinfo | he5_swaliasinfo | Retrieves information about field aliases | 2-48 |
| | HE5_SWwritefield | he5_swwrfl | Writes data to a swath field | 2-136 |
| | HE5_SWreadfield | he5_swrdfl | Reads data from a swath field. | 2-118 |
| | HE5_SWwriteattr | he5_swwrattr | Writes/updates attribute in a swath | 2-132 |
| | HE5_SWreadattr | he5_swrdattr | Reads attribute from a swath | 2-116 |
| | HE5_SWwritegrpattr | he5_swwrgattr | Writes/updates group attribute in a swath | 2-141 |
| | HE5_SWwritelocattr | he5_swrlattr | Write/updates group attribute in a swath | 2-143 |
| | HE5_SWreadgrpatr | he5_swrdgattr | Reads attribute from a swath | 2-120 |
| | HE5_SWreadlocattr | he5_swrdlatr | Reads attribute from a swath | 2-121 |
| | HE5_SWsetfillvalue | he5_swsetfill | Sets fill value for the specified field | 2-128 |
| Information | HE5_SWgetfillvalue | he5_swgetfill | Retrieves fill value for the specified field | 2-91 |
| | HE5_SWinqdims | he5_swinqdims | Retrieves information about dimensions defined in swath | 2-97 |
| | HE5_SWinqmaps | he5_swinqmaps | Retrieves information about the geolocation relations defined | 2-103 |
| | HE5_SWinqidxmaps | he5_swinqimaps | Retrieves information about the indexed geolocation/data mappings defined | 2-100 |
| | HE5_SWinqgeofields | he5_swinqgfls | Retrieves information about the geolocation fields defined | 2-98 |
| | HE5_SWinqdatafields | he5_swinqdflds | Retrieves information about the data fields defined | 2-96 |
| Attributes | HE5_SWinqattrs | he5_swinqatrs | Retrieves number and names of attributes defined | 2-95 |

Table 1-2. Summary of the Swath Interface (2 of 3)

| Category | Routine Name | | Description | Page Nos. |
|----------|---------------------|-------------------|---|-----------|
| | C | FORTRAN | | |
| Inquiry | HE5_SWinqgrpattrs | he5_swinqgattr | Retrieve information about group attributes defined in swath | 2-99 |
| | HE5_SWinqlocattrs | he5_swinqlattr | Retrieve information about local attributes defined in swath | 2-101 |
| | HE5_SWlocattrinfo | he5_swlocattrinfo | Returns information about a data field's local attribute(s) | 2-107 |
| | HE5_SWnentries | he5_swnentries | Returns number of entries and descriptive string buffer size for a specified entity | 2-111 |
| | HE5_SWdiminfo | he5_swdiminfo | Retrieve size of specified dimension | 2-78 |
| | HE5_SWmapinfo | he5_swmapinfo | Retrieve offset and increment of specified geolocation mapping | 2-108 |
| | HE5_SWidxmapinfo | he5_swimapinfo | Retrieve offset and increment of specified geolocation mapping | 2-94 |
| Swath | HE5_SWattrinfo | he5_swatrinfo | Returns information about swath attributes | 2-51 |
| | HE5_SWgrpattrinfo | he5_swgattrinfo | Returns information about a swath group attribute | 2-93 |
| | HE5_SWfieldinfo | he5_swfldinfo | Retrieve information about a specific geolocation or data field | 2-86 |
| | HE5_SWcompinfo | he5_swcompinfo | Retrieve compression information about a field | 2-53 |
| | HE5_SWinqswath | he5_swinqswath | Retrieves number and names of swaths in file | 2-105 |
| | HE5_SWregionindex | he5_swregidx | Returns information about the swath region ID | 2-122 |
| | HE5_SWupdateidxmap | he5_swupimap | Update map index for a specified region | 2-130 |
| | HE5_SWgeomapinfo | he5_swgmapinfo | Retrieve type of dimension mapping for a dimension | 2-92 |
| Subset | HE5_SWdefboxregion | he5_swdefboxreg | Define region of interest by latitude/longitude | 2-55 |
| | HE5_SWregioninfo | he5_swreginfo | Returns information about defined region | 2-124 |
| | HE5_SWextractregion | he5_swextreg | Read a region of interest from a field | 2-84 |
| | HE5_SWdeftimeperiod | he5_swdeftmeper | Define a time period of interest | 2-72 |
| | HE5_SWperiodinfo | he5_swperinfo | Retuns information about a defined time period | 2-114 |
| | HE5_SWextractperiod | he5_swextper | Extract a defined time period | 2-82 |
| | HE5_SWdefvrtregion | he5_swdefvrtreg | Define a region of interest by vertical field | 2-74 |
| | HE5_SWdupregion | he5_swdupreg | Duplicate a region or time period | 2-80 |
| Profile | HE5_PRdefine | he5_prdefine | Defines profile data structure | 2-145 |
| | HE5_PRread | he5_prread | Reads profile data | 2-150 |
| | HE5_PRwrite | he5_prwrite | Writes profile data | 2-153 |
| | HE5_PRinquire | he5_prinquire | Retrieves information about profiles | 2-149 |
| | HE5_PRinfo | he5_prinfo | Return information about profile | 2-147 |
| | HE5_PRreclaimspace | Not available | Reclaims memory used by data buffer in HE5_PRread()call | 2-152 |

Table 1-2. Summary of the Swath Interface (3 of 3)

| Category | Routine Name | | Description | Page Nos. |
|--------------------|---------------------|---------------|-----------------------------|-----------|
| | C | FORTRAN | | |
| External Files | HE5_SWmountexternal | Not available | Mount external data file | 2-110 |
| | HE5_SWreadexternal | Not available | Read external data set | 2-117 |
| | HE5_SWunmount | Not available | Dismount external data file | 2-129 |
| External Data Sets | HE5_SWsetextdata | he5_swsetxdat | Set external data set | 2-127 |
| Data Sets | HE5_SWgetextdata | he5_swgetxdat | Get external data set | 2-89 |

1.5 Grid Data

The GD (*Grid*) interface consists of routines for storing, retrieving, and manipulating data in grid data sets. This interface is designed to support data that has been stored in a rectilinear array based on a well defined and explicitly supported projection. See the Users' Guide, Volume 1 that accompanies this document for more details.

1.5.1 The Grid Data Interface

All C routine names in the grid data interface have the prefix “HE5_GD” and the equivalent FORTRAN routine names are prefixed by “he5_gd.” The GD routines are classified into the following categories:

- **Access routines** initialize and terminate access to the GD interface and grid data sets (including opening and closing files).
- **Definition routines** allow the user to set key features of a grid data set.
- **Basic I/O routines** read and write data and metadata to a grid data set.
- **Inquiry routines** return information about data contained in a grid data set.
- **Subset routines** allow reading of data from a specified geographic region.

1.5.2 List of Grid API Routines

The GD function calls are listed below in Table 1-3 and are described in detail in Section 2 of this document. The listing in Section 2 is in alphabetical order.

Table 1-3. Summary of the Grid Interface (1 of 3)

| Category | Routine Name | | Description | Page Nos. |
|----------|-----------------|-----------------|---|-----------|
| | C | FORTRAN | | |
| Access | HE5_GDopen | he5_gdopen | Creates a new file or opens an existing one | 2-211 |
| | HE5_GDcreate | he5_gdcreate | Creates a new grid in the file | 2-162 |
| | HE5_GDattach | he5_gdattach | Attaches to a grid | 2-156 |
| | HE5_GDdetach | he5_gddetach | Detaches from grid interface | 2-185 |
| | HE5_GDclose | he5_gdclose | Closes file | 2-160 |
| | HE5_GDdeforigin | he5_gddeforigin | Defines origin of grid | 2-173 |
| | HE5_GDdefdim | he5_gddefdim | Defines dimensions for a grid | 2-170 |
| | HE5_GDdefproj | he5_gddefproj | Defines projection of grid | 2-175 |

Table 1-3. Summary of the Grid Interface (2 of 3)

| Category | Routine Name | | Description | Page Nos. |
|------------|----------------------|------------------|---|-----------|
| | C | FORTRAN | | |
| Definition | HE5_GDdefpixreg | he5_gddefpixreg | Defines pixel registration within grid cell | 2-174 |
| | HE5_GDdeffield | he5_gddeffld | Defines data fields to be stored in a grid | 2-171 |
| | HE5_GDdefcomp | he5_gddefcomp | Defines a field compression scheme | 2-166 |
| | HE5_GDblkSOMoffset | None | This is a special function for SOM MISR data. Write block SOM offset values. | 2-158 |
| | HE5_GDdefcomtile | he5_gddefcomtile | Defines compression with automatic tiling | 2-168 |
| Basic I/O | HE5_GDwritefieldmeta | he5_gdwrmeta | Writes metadata for field already existing in file | 2-229 |
| | HE5_GDwritefield | he5_gdwrfld | Writes data to a grid field. | 2-227 |
| | HE5_GDreadfield | he5_gdrdfld | Reads data from a grid field | 2-217 |
| | HE5_GDwriteattr | he5_gdwrattr | Writes/updates attribute in a grid. | 2-225 |
| | HE5_GDwritelocattr | he5_gdwrllattr | Writes/updates group attribute in a grid | 2-232 |
| | HE5_GDwritegrpattr | he5_gdwrgattr | Writes/updates attribute in a grid | 2-230 |
| | HE5_GDreadattr | he5_gdrdattr | Reads attribute from a grid | 2-216 |
| | HE5_GDreadgrpattr | he5_gdrdgattr | Reads attribute from a grid | 2-219 |
| | HE5_GDreadlocattr | he5_gdrllattr | Reads attribute from a grid | 2-220 |
| | HE5_GDsetfillvalue | he5_gdsetfill | sets fill value for the specified field | 2-224 |
| | HE5_GDgetfillvalue | he5_gdgetfill | Retrieves fill value for the specified field | 2-193 |
| Inquiry | HE5_GDinqdims | he5_gdinqdims | Retrieves information about dimensions defined in grid | 2-201 |
| | HE5_GDinqfields | he5_gdinqflds | Retrieves information about the data fields defined in grid | 2-202 |
| | HE5_GDinqattrs | he5_gdinqattrs | Retrieves number and names of attributes defined | 2-200 |
| | HE5_GDinqlocattrs | he5_gdinql attrs | Retrieve information about local attributes defined for a field | 2-205 |
| | HE5_GDinqgrpattrs | he5_gdinqg attrs | Retrieve information about group attributes defined in grid | 2-204 |
| | HE5_GDnentries | he5_gdnentries | Returns number of entries and descriptive string buffer size for a specified entity | 2-210 |
| | HE5_GDgridinfo | he5_gdgridinfo | Returns dimensions of grid and X-Y coordinates of corners | 2-198 |
| | HE5_GDprojinfo | he5_gdprojinfo | Returns all GCTP projection information | 2-215 |
| | HE5_GDdiminfo | he5_gddiminfo | Retrieves size of specified dimension. | 2-186 |
| | HE5_GDcompinfo | he5_gdcompinfo | Retrieve compression information about a field | 2-161 |
| | HE5_GDfieldinfo | he5_gdfldinfo | Retrieves information about a specific geolocation or data field in the grid | 2-189 |
| | HE5_GDinqgrid | he5_gdinqgrid | Retrieves number and names of grids in file | 2-203 |
| | HE5_GDattrinfo | he5_gdattrinfo | Returns information about grid attributes | 2-157 |

Table 1-3. Summary of the Grid Interface (3 of 3)

| Category | Routine Name | | Description | Page Nos. |
|-----------|---------------------|-------------------|---|-----------|
| | C | FORTRAN | | |
| | HE5_GDgrpattrinfo | he5_gdgattrinfo | Returns information about a swath group attribute | 2-199 |
| | HE5_GDlocattrinfo | he5_gdlattrinfo | Returns information about a Data Field's local attribute(s) | 2-209 |
| | HE5_GDorigininfo | he5_gdorginfo | Return information about grid origin | 2-213 |
| | HE5_GDpixreginfo | he5_gdpreginfo | Return pixel registration information for given grid | 2-214 |
| | HE5_GDdefboxregion | he5_gddefboxreg | Define region of interest by latitude/longitude | 2-165 |
| | HE5_GDregioninfo | he5_gdreginfo | Returns information about a defined region | 2-221 |
| | HE5_GDextractregion | he5_gdextrreg | Read a region of interest from a field | 2-188 |
| Subset | HE5_GDdeftimeperiod | he5_gddeftmep | Define a time period of interest | 2-180 |
| | HE5_GDdefvrtregion | he5_gddefvrtreg | Define a region of interest by vertical field | 2-182 |
| | HE5_GDgetpixels | he5_gdgetpix | get row/columns for lon/lat pairs | 2-194 |
| | HE5_GDgetpixvalues | he5_gdgetpixval | get field values for specified pixels | 2-196 |
| | HE5_GDinterpolate | he5_gdinterpolate | Perform bilinear interpolation on a grid field | 2-207 |
| | HE5_GDdupregion | he5_gddupreg | Duplicate a region or time period | 2-187 |
| Tiling | HE5_GDdeftile | he5_gddeftle | Define a tiling scheme | 2-177 |
| External | HE5_GDsetxdata | he5_gdsetxdat | Set external data set | 2-223 |
| Data Sets | HE5_GDgetxdata | he5_gdgetxdat | Get external data set | 2-191 |

1.6 GCTP Usage

The HDF-EOS Grid API uses the U.S. Geological Survey General Cartographic Transformation Package (GCTP) to define and subset grid structures. This section described codes used by the package.

1.6.1 GCTP Projection Codes

The following GCTP projection codes are used in the grid API described in Section 4 below:

| | | |
|-----------------|------|--------------------------------------|
| HE5_GCTP_ALBERS | | Albers Conical Equal-Area Projection |
| HE5_GCTP_MERCAT | | Mercator Projection |
| HE5_GCTP_SPCS | | State Plane Coordinate System |
| HE5_GCTP_GEO | (0) | Geographic |
| HE5_GCTP_UTM | (1) | Universal Transverse Mercator |
| HE5_GCTP_LAMCC | (4) | Lambert Conformal Conic |
| HE5_GCTP_PS | (6) | Polar Stereographic |
| HE5_GCTP_POLYC | (7) | Polyconic |
| HE5_GCTP_TM | (9) | Transverse Mercator |
| HE5_GCTP_LAMAZ | (11) | Lambert Azimuthal Equal Area |
| HE5_GCTP_HOM | (20) | Hotine Oblique Mercator |
| HE5_GCTP_SOM | (22) | Space Oblique Mercator |
| HE5_GCTP_GOOD | (24) | Interrupted Goode Homolosine |

* The Integerized Sinusoidal Projection is not part of the original GCTP package. It has been added by ECS. See *Level-3 SeaWiFS Data Products: Spatial and Temporal Binning Algorithms*. Additional references are provided in Section 2.

Note that other projections supported by GCTP will be adapted for HDF-EOS Version 5 as new user requirements are surfaced. For further details on the GCTP projection package, please refer to Section 6.3.4 and Appendix G of the SDP Toolkit Users Guide for the ECS Project, November 2000, (333-CD-600-001).

1.6.2 UTM Zone Codes

The Universal Transverse Mercator (UTM) Coordinate System uses zone codes instead of specific projection parameters. The table that follows lists UTM zone codes as used by GCTP Projection Transformation Package. C.M. is Central Meridian

| Zone | C.M. | Range | Zone | C.M. | Range |
|------|------|-----------|------|------|-----------|
| 01 | 177W | 180W-174W | 31 | 003E | 000E-006E |
| 02 | 171W | 174W-168W | 32 | 009E | 006E-012E |
| 03 | 165W | 168W-162W | 33 | 015E | 012E-018E |
| 04 | 159W | 162W-156W | 34 | 021E | 018E-024E |
| 05 | 153W | 156W-150W | 35 | 027E | 024E-030E |
| 06 | 147W | 150W-144W | 36 | 033E | 030E-036E |
| 07 | 141W | 144W-138W | 37 | 039E | 036E-042E |
| 08 | 135W | 138W-132W | 38 | 045E | 042E-048E |
| 09 | 129W | 132W-126W | 39 | 051E | 048E-054E |
| 10 | 123W | 126W-120W | 40 | 057E | 054E-060E |
| 11 | 117W | 120W-114W | 41 | 063E | 060E-066E |
| 12 | 111W | 114W-108W | 42 | 069E | 066E-072E |
| 13 | 105W | 108W-102W | 43 | 075E | 072E-078E |
| 14 | 099W | 102W-096W | 44 | 081E | 078E-084E |
| 15 | 093W | 096W-090W | 45 | 087E | 084E-090E |
| 16 | 087W | 090W-084W | 46 | 093E | 090E-096E |
| 17 | 081W | 084W-078W | 47 | 099E | 096E-102E |
| 18 | 075W | 078W-072W | 48 | 105E | 102E-108E |
| 19 | 069W | 072W-066W | 49 | 111E | 108E-114E |
| 20 | 063W | 066W-060W | 50 | 117E | 114E-120E |
| 21 | 057W | 060W-054W | 51 | 123E | 120E-126E |
| 22 | 051W | 054W-048W | 52 | 129E | 126E-132E |
| 23 | 045W | 048W-042W | 53 | 135E | 132E-138E |
| 24 | 039W | 042W-036W | 54 | 141E | 138E-144E |
| 25 | 033W | 036W-030W | 55 | 147E | 144E-150E |
| 26 | 027W | 030W-024W | 56 | 153E | 150E-156E |
| 27 | 021W | 024W-018W | 57 | 159E | 156E-162E |
| 28 | 015W | 018W-012W | 58 | 165E | 162E-168E |
| 29 | 009W | 012W-006W | 59 | 171E | 168E-174E |
| 30 | 003W | 006W-000E | 60 | 177E | 174E-180W |

1.6.3 GCTP Spheroid Codes

| | |
|-----------------------|--------|
| Clarke 1866 (default) | (0) |
| Clarke 1880 | (1) |
| Bessel | (2) |
| International 1967 | (3) |
| International 1909 | (4) |
| WGS 72 | (5) |
| Everest | (6) |
| WGS 66 | (7) |
| GRS 1980 | (8) |
| Airy | (9) |
| Modified Airy | (10) |

| | |
|---------------------------|------|
| Modified Everest | (11) |
| WGS 84 | (12) |
| Southeast Asia | (13) |
| Australlian National | (14) |
| Krassovsky | (15) |
| Hough | (16) |
| Mercury 1960 | (17) |
| Modified Mercury 1968 | (18) |
| Sphere of Radius 6370997m | (19) |

1.6.4 GCTP Projection Parameters

Table 1-4. Projection Transformation Package Projection Parameters (1 of 2)

| Code & Projection Id | Array Element | | | | | | | |
|---------------------------|---------------|--------|--------|--------|---------|-----------|----|----|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 0 Geographic | | | | | | | | |
| 1 UTM | Lon/Z | Lat/Z | | | | | | |
| 4 Lambert Conformal C | SMajor | SMinor | STDPR1 | STDPR2 | CentMer | OriginLat | FE | FN |
| 6 Polar Stereographic | SMajor | SMinor | | | LongPol | TrueScale | FE | FN |
| 7 Polyconic | SMajor | SMinor | | | CentMer | OriginLat | FE | FN |
| 9 Transverse Mercator | SMajor | SMinor | Factor | | CentMer | OriginLat | FE | FN |
| 11 Lambert Azimuthal | Sphere | | | | CentLon | CenterLat | FE | FN |
| 20 Hotin Oblique Merc A | SMajor | SMinor | Factor | | | OriginLat | FE | FN |
| 20 Hotin Oblique Merc B | SMajor | SMinor | Factor | AziAng | AzmthPt | OriginLat | FE | FN |
| 22 Space Oblique Merc A | SMajor | SMinor | | IncAng | AscLong | | FE | FN |
| 22 Space Oblique Merc B | SMajor | SMinor | Satnum | Path | | | FE | FN |
| 24 Interrupted Goode | Sphere | | | | | | | |
| 99 Integerized Sinusoidal | Sphere | | | | CentMer | | FE | FN |

Table 1-4. Projection Transformation Package Projection Parameters (2 of 2)

| Code & Projection Id | Array Element | | | | |
|---------------------------|---------------|------|-------|---------------------|------|
| | 9 | 10 | 11 | 12 | 13 |
| 0 Geographic | | | | | |
| 1 UTM | | | | | |
| 4 Lambert Conformal C | | | | | |
| 6 Polar Stereographic | | | | | |
| 7 Polyconic | | | | | |
| 9 Transverse Mercator | | | | | |
| 11 Lambert Azimuthal | | | | | |
| 20 Hotin Oblique Merc A | Long1 | Lat1 | Long2 | Lat2 | Zero |
| 20 Hotin Oblique Merc B | | | | | One |
| 22 Space Oblique Merc A | PSRev | Srat | PFlag | HDF-EOS Variable | Zero |
| 22 Space Oblique Merc B | | | | HDF-EOS Variable | One |
| 24 Interrupted Goode | | | | | |
| 99 Integerized Sinusoidal | Nzone | | RFlag | | |

Where,

- Lon/Z Longitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
- Lat/Z Latitude of any point in the UTM zone or zero. If zero, a zone code must be specified.
- Smajor Semi-major axis of ellipsoid. If zero, Clarke 1866 in meters is assumed.
- Sminor Eccentricity squared of the ellipsoid if less than zero, if zero, a spherical form is assumed, or if greater than zero, the semi-minor axis of ellipsoid.
- Sphere Radius of reference sphere. If zero, 6370997 meters is used.
- STDPR1 Latitude of the first standard parallel

| | |
|-----------|--|
| STDPR2 | Latitude of the second standard parallel |
| CentMer | Longitude of the central meridian |
| OriginLat | Latitude of the projection origin |
| FE | False easting in the same units as the semi-major axis |
| FN | False northing in the same units as the semi-major axis |
| TrueScale | Latitude of true scale |
| LongPol | Longitude down below pole of map |
| Factor | Scale factor at central meridian (Transverse Mercator) or center of projection (Hotine Oblique Mercator) |
| CentLon | Longitude of center of projection |
| CenterLat | Latitude of center of projection |
| Long1 | Longitude of first point on center line (Hotine Oblique Mercator, format A) |
| Long2 | Longitude of second point on center line (Hotine Oblique Mercator, frmt A) |
| Lat1 | Latitude of first point on center line (Hotine Oblique Mercator, format A) |
| Lat2 | Latitude of second point on center line (Hotine Oblique Mercator, format A) |
| AziAng | Azimuth angle east of north of center line (Hotine Oblique Mercator, frmt B) |
| AzmthPt | Longitude of point on central meridian where azimuth occurs (Hotine Oblique Mercator, format B) |
| IncAng | Inclination of orbit at ascending node, counter-clockwise from equator (SOM, format A) |
| AscLong | Longitude of ascending orbit at equator (SOM, format A) |
| PSRev | Period of satellite revolution in minutes (SOM, format A) |
| SRat | Satellite ratio to specify the start and end point of x,y values on earth surface (SOM, format A -- for Landsat use 0.5201613) |
| PFlag | End of path flag for Landsat: 0 = start of path, 1 = end of path (SOM, frmt A) |
| Satnum | Landsat Satellite Number (SOM, format B) |
| Path | Landsat Path Number (Use WRS-1 for Landsat 1, 2 and 3 and WRS-2 for Landsat 4 and 5.) (SOM, format B) |

| | |
|-------|---|
| Nzone | Number of equally spaced latitudinal zones (rows); must be two or larger and even |
| Rflag | Right justify columns flag is used to indicate what to do in zones with an odd number of columns. If it has a value of 0 or 1, it indicates the extra column is on the right (zero) or left (one) of the projection Y-axis. If the flag is set to 2 (two), the number of columns are calculated so there are always an even number of columns in each zone. |

Notes:

- HDF-EOS variable is used by the library function HE5_GDblksomoffset.
- Array elements 14 and 15 are set to zero.
- All array elements with blank fields are set to zero.

All angles (latitudes, longitudes, azimuths, etc.) are entered in packed degrees/ minutes/ seconds (DDDDMMMSSS.SS) format.

The following notes apply to the Space Oblique Mercator A projection:

- A portion of Landsat rows 1 and 2 may also be seen as parts of rows 246 or 247. To place these locations at rows 246 or 247, set the end of path flag (parameter 11) to 1--end of path. This flag defaults to zero.
- When Landsat-1,2,3 orbits are being used, use the following values for the specified parameters:
 - Parameter 4 099005031.2
 - Parameter 5 128.87 degrees - (360/251 * path number) in packed DMS format
 - Parameter 9 103.2669323
 - Parameter 10 0.5201613
- When Landsat-4,5 orbits are being used, use the following values for the specified parameters:
 - Parameter 4 098012000.0
 - Parameter 5 129.30 degrees - (360/233 * path number) in packed DMS format
 - Parameter 9 98.884119
 - Parameter 10 0.5201613

2. Function Reference

2.1 Format

This section contains a function-by-function reference for each interface in the HDF-EOS library. Each function has a separate page describing it (in some cases there are multiple pages). Each page contains the following information (in order):

- Function name as used in C
- Function declaration in ANSI C format
- Description of each argument
- Purpose of routine
- Description of returned value
- Description of the operation of the routine
- A short example of how to use the routine in C
- The FORTRAN declaration of the function and arguments
- An equivalent FORTRAN example

2.1.1 Point Interface Functions

This section contains an alphabetical listing of all the functions in the Point interface. The functions are alphabetized based on their C-language names.

Attach to an Existing Point Structure

HE5_PTattach

hid_t HE5_PTattach(hid_t *fid*, const char **pointname*)

| | |
|------------------|---|
| <i>fid</i> | IN: Point file ID returned by HE5_PTopen |
| <i>pointname</i> | IN: Name of point to be attached |
| Purpose | Attaches to an existing point within the file. |
| Return value | Returns the point handle (pointID) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point file ID or point name. |
| Description | This routine attaches to the point using the <i>pointname</i> parameter as the identifier. |
| Example | In this example, we attach to the previously created point, "ExamplePoint", within the HDF-EOS file, <i>Point.he5</i> , referred to by the handle, <i>fid</i> : |

```
pointID = HE5_PTattach(fid, "ExamplePoint");
```

The point can then be referenced by subsequent routines using the handle, *pointID*.

FORTRAN integer function he5_ptattach(*fid*,*pointname*)
 integer *fid*
 character*(*) *pointname*

The equivalent *FORTRAN* code for the example above is:

```
pointid = he5_ptattach(fid, "ExamplePoint")
```

Return Information About a Point Attribute

HE5_PTattrinfo

```
herr_t HE5_PTattrinfo(hid_t pointID, const char *attrname, H5T_class_t *  
                      numbertype, hsize_t *count)
```

| | |
|-------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: Attribute name |
| <i>numbertype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of total bytes in attribute |
| Purpose | Returns information about a point attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a point attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_PTattrinfo(pointID, "ScalarFloat",&nt,&count);
```

The *nt* variable will have the value 1 and *count* will have the value 4.

FORTRAN integer function he5_ptattrinfo(*pointid*,*attrname*,*ntype*,*count*)

```
integer      pointid  
character *(*) attrname  
integer      ntype  
integer*4    count
```

The equivalent *FORTRAN* code for the example above is:

```
pointid = he5_ptattrinfo(pointid, "ScalarFloat",ntype,count)
```

Return Linkage Field to Previous Level

HE5_PTbcklinkinfo

herr_t HE5_PTbcklinkinfo(hid_t *pointID*, int *level*, char **linkfield*)

| | | |
|------------------|------|--|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: | Point level (0-based) |
| <i>linkfield</i> | OUT: | Link field |
| Purpose | | Returns the linkfield to the previous level. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | This routine returns the linkfield to the previous level. |
| Example | | In this example, we return the linkfield connecting the <i>Observations</i> level to the previous <i>Desc-Loc</i> level. (This levels are defined in the <i>HE5_PTdeflevel</i> routine.) |

```
status = HE5_PTbcklinkinfo(pointID2, 1, linkfield);
```

The *linkfield* will contain the string: *ID*.

FORTRAN integer function he5_ptbcklinkinfo(*pointid*,*level*,*linkfield*)

```
integer      pointid,status  
character *(*) linkfield  
integer      level
```

The equivalent *FORTRAN* code for the example above is:

```
level = 1  
  
status = he5_ptbcklinkinfo(pointid, level, linkfield)
```

Close an HDF-EOS File

HE5_PTclose

herr_t HE5_PTclose(hid_t *fid*)

fid IN: Point file ID returned by HE5_PTopen

Purpose Closes file.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This routine closes the HDF-EOS Point file.

Example

```
status = HE5_PTclose(fid);
```

FORTRAN integer function he5_ptclose(*fid*)

```
integer fid
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptclose(fid)
```

Create a New Point Structure

HE5_PTcreate

hid_t HE5_PTcreate(hid_t *fid*, const char **pointname*)

| | |
|------------------|--|
| <i>fid</i> | IN: Point file ID returned by HE5_PTopen |
| <i>pointname</i> | IN: Name of point to be created |
| Purpose | Creates a point within the file. |
| Return value | Returns the point handle (<i>pointID</i>) if successful or FAIL (-1) otherwise. |
| Description | The point is created as a Compound dataset within the HDF-EOS file with the name <i>pointname</i> . |
| Example | In this example, we create a new point structure, <i>ExamplePoint</i> , in the previously created file, <i>Point.he5</i> . |

```
pointID = HE5_PTcreate(fid, "ExamplePoint");
```

The point structure is then referenced by subsequent routines using the handle, *pointID*.

FORTRAN integer function he5_ptcreate(*fid*,*pointname*)
 integer *pointid, fid*
 character *(*) *pointame*

The equivalent *FORTRAN* code for the example above is:

```
pointid = he5_ptcreate(fid, "ExamplePoint")
```

Define a New Level Within a Point

HE5_PTdeflevel

`herr_t HE5_PTdeflevel(hid_t pointID, const char *levelname, HE5_CmpDTSinfo *levelinfo)`

| | |
|------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>levelname</i> | IN: Name of level to be defined |
| <i>levelinfo</i> | IN: C-data structure containing all necessary information about level to be defined |
| | Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,” |
| Purpose | Defines a new level within the point. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine defines a level within the point. A simple point consists of a single level. A point where there is common data for a number of records can be more efficiently stored with multiple levels. The order in which the levels are defined determines the (0-based) level index. |
| Example | <u>Simple Point</u> |

In this example, we define a simple single level point, with *levelname*, *Sensor*. The *levelname* should not contain any slashes (“/”). It consists of six fields, *ID*, *Time*, *Longitude*, *Latitude*, *Temperature*, and *Mode* defined in the field list. The *fieldtype* and *fieldorder* parameters are arrays consisting of the HDF number type codes and field orders, respectively. The *Temperature* is an array field of dimension 4 and the *Mode* field a character string of size 4. All other fields are scalars. Note that the order for numerical scalar variables can be either 0 or 1.

```
typedef struct {

    int      id;
    int      time;
    float   lon;
    float   lat;
    float   temp[4];
    char    mode[4];
```

```

} InputData1;

HE5_CmpDTSinfo  dtsinfo;
dtsinfo.nfields = 6;
dtsinfo.rank[0] = 1;
dtsinfo.rank[1] = 1;
dtsinfo.rank[2] = 1;
dtsinfo.rank[3] = 1;
dtsinfo.rank[4] = 1;
dtsinfo.rank[5] = 1;
for (i = 0; i < 6; i++)
    dtsinfo.fieldname[i] = (char
*)malloc(64,sizeof(char));

strcpy(dtsinfo.fieldname[0], "ID");
strcpy(dtsinfo.fieldname[1], "Time");
strcpy(dtsinfo.fieldname[2], "Longitude");
strcpy(dtsinfo.fieldname[3], "Latitude");
strcpy(dtsinfo.fieldname[4], "Temperature");
strcpy(dtsinfo.fieldname[5], "Mode");

dtsinfo.offset[0] = HOFFSET(InputData1, id);
dtsinfo.offset[1] = HOFFSET(InputData1, time);
dtsinfo.offset[2] = HOFFSET(InputData1, lon);
dtsinfo.offset[3] = HOFFSET(InputData1, lat);
dtsinfo.offset[4] = HOFFSET(InputData1, temp);
dtsinfo.offset[5] = HOFFSET(InputData1, mode);

dtsinfo.datatype[0] = H5T_NATIVE_INT;
dtsinfo.datatype[1] = H5T_NATIVE_INT;
dtsinfo.datatype[2] = H5T_NATIVE_FLOAT;

```

```
dtsinfo.datatype[3] = H5T_NATIVE_FLOAT;
dtsinfo.datatype[4] = H5T_NATIVE_FLOAT;
dtsinfo.datatype[5] = H5T_NATIVE_CHAR;

dtsinfo.dims[0][0] = 1;
dtsinfo.dims[1][0] = 1;
dtsinfo.dims[2][0] = 1;
dtsinfo.dims[3][0] = 1;
dtsinfo.dims[4][0] = 4;
dtsinfo.dims[5][0] = 4;

dtsinfo.datasize = sizeof(InputData1);
status = HE5_PTdeflevel(pointID1, "Sensor", &dtsinfo);
for (i = 0; i < 6; i++)
    free(dtsinfo.fieldname[i]);
```

Multi-Level Point

In this example, we define a two-level point that describes data from a network of fixed buoys. The first level contains information about each buoy and includes the name (label) of the buoy, its (fixed) longitude and latitude, its deployment date, and an ID that is used to link it to the following level. (The link field is defined in the *HE5_PTdeflinkage* routine described later.) The entries within this ID field must be unique. The second level contains the actual measurements from the buoys (rainfall and temperature values) plus the observation time and the ID which relates a given measurement to a particular buoy entry in the previous level. There can be many records in this level with the same ID since there can be multiple measurements from a single buoy. It is advantageous, although not mandatory, to store all records for a particular buoy (ID) contiguously.

Level 0

```
HE5_CmpDTSinfo    lev0_info;

typedef struct {

    char      label[8];

    double    lon;

    double    lat;

    float     deploydate;

    char      id;

} Lev0_Data;

lev0_info.nfields = 5;

lev0_info.rank[0] = 1;

lev0_info.rank[1] = 1;

lev0_info.rank[2] = 1;

lev0_info.rank[3] = 1;

lev0_info.rank[4] = 1;

for (i = 0; i < 5; i++)

    lev0_info.fieldname[i] = (char
*)calloc(64,sizeof(char));

strcpy(lev0_info.fieldname[0], "Label");
```

```

strcpy(lev0_info.fieldname[1], "Longitude");
strcpy(lev0_info.fieldname[2], "Latitude");
strcpy(lev0_info.fieldname[3], "DeployDate");
strcpy(lev0_info.fieldname[4], "ID");

lev0_info.offset[0] = HOFFSET(Lev0_Data, label);
lev0_info.offset[1] = HOFFSET(Lev0_Data, lon);
lev0_info.offset[2] = HOFFSET(Lev0_Data, lat);
lev0_info.offset[3] = HOFFSET(Lev0_Data, deploydate);
lev0_info.offset[4] = HOFFSET(Lev0_Data, id);

lev0_info.datatype[0] = H5T_NATIVE_CHAR;
lev0_info.datatype[1] = H5T_NATIVE_DOUBLE;
lev0_info.datatype[2] = H5T_NATIVE_DOUBLE;
lev0_info.datatype[3] = H5T_NATIVE_FLOAT;
lev0_info.datatype[4] = H5T_NATIVE_CHAR;

lev0_info.dims[0][0] = 8;
lev0_info.dims[1][0] = 1;
lev0_info.dims[2][0] = 1;
lev0_info.dims[3][0] = 1;
lev0_info.dims[4][0] = 1;

lev0_info.datasize = sizeof(Lev0_Data);

status = HE5_PTdeflevel(pointID2, "Desc-Loc", &lev0_info);
for (i = 0; i < 5; i++)
    free (lev0_info.fieldname[i]);

```

Level 1

```
HE5_CmpDTSinfo    lev1_info;
```

```

typedef struct {

    double    time;
    float     rain;
    float     temp;
    char      id;

} Lev1_Data;

lev1_info.nfields = 4;
lev1_info.rank[0] = 1;
lev1_info.rank[1] = 1;
lev1_info.rank[2] = 1;
lev1_info.rank[3] = 1;
for (i = 0; i < 4; i++)

    lev1_info.fieldname = (char *)calloc(64,sizeof(char));

strcpy(lev1_info.fieldname[0], "Time");
strcpy(lev1_info.fieldname[1], "Rainfall");
strcpy(lev1_info.fieldname[2], "Temperature");
strcpy((lev1_info.fieldname[3], "ID"));

lev1_info.offset[0] = HOFFSET(Lev1_Data, time);
lev1_info.offset[1] = HOFFSET(Lev1_Data, rain);
lev1_info.offset[2] = HOFFSET(Lev1_Data, temp);
lev1_info.offset[3] = HOFFSET(Lev1_Data, id);

lev1_info.datatype[0] = H5T_NATIVE_DOUBLE;
lev1_info.datatype[1] = H5T_NATIVE_FLOAT;
lev1_info.datatype[2] = H5T_NATIVE_FLOAT;
lev1_info.datatype[3] = H5T_NATIVE_CHAR;

lev1_info.dims[0][0] = 1;

```

```
levl_info.dims[1][0] = 1;  
levl_info.dims[2][0] = 1;  
levl_info.dims[3][0] = 1;  
  
levl_info.datasize = sizeof(Levl_Data);  
  
status = HE5_PTdeflevel(pointID2, "Observations",  
&levl_info);  
  
for (i = 0; i < 4; i++)  
    free(levl_info.fieldname[i]);
```

FORTRAN See Example 2 from Section 7.1.1.2 of Volume 1 (Overview and Examples)

Define Linkage Field Between Two Levels

HE5_PTdeflinkage

```
herr_t HE5_PTdeflinkage(hid_t pointID, char *parent, char *child, char  
*linkfield)
```

| | |
|------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>parent</i> | IN: Name of parent level |
| <i>child</i> | IN: Name of child level |
| <i>linkfield</i> | IN: Name of (common) link field |
| Purpose | Defines a link field between two (adjacent) levels. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine defines the link field between two levels. This field must be defined in both levels. |
| Note | The defining of a linkage is necessary if more than one level is defined. |
| Example | In this example we define the <i>ID</i> field as the link between the two levels defined previously in the <i>HE5_PTdeflevel</i> routine. |

```
status = HE5_PTdeflinkage(pointID2, "Desc-Loc",  
"Observations", "ID");
```

FORTRAN

```
integer function  
he5_ptdeflinkage(pointid,levelname1,levelname2,linkname)  
integer      pointid,status  
character *(*) linkname,levelname1,levelname2
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptdeflinkage(pointid, "Desc-Loc",  
"Observations", "ID")
```


Detach from Point Structure

HE5_PTdetach

herr_t HE5_PTdetach(hid_t *pointID*)

| | |
|----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| Purpose | Detaches from point data set. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine should be run before exiting from the point file for every point opened by <i>HE5_PTcreate</i> or <i>HE5_PTattach</i> . |
| Example | In this example, we detach the point structure, <i>ExamplePoint</i> : |

```
status = HE5_PTdetach(pointID);
```

FORTRAN integer function he5_ptdetach(*pointid*)
 integer *pointid,status*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptdetach(pointid)
```

Return Linkage Field to Following Level

HE5_PTfwdlinkinfo

herr_t HE5_PTfwdlinkinfo(hid_t *pointID*, int *level*, char **linkfield*)

| | | |
|------------------|------|---|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: | Point level (0-based) |
| <i>linkfield</i> | OUT: | Link field |
| Purpose | | Returns the link field to the following level. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | This routine returns the link field to the following level. |
| Example | | In this example, we return the link field connecting the <i>Desc-Loc</i> level to the following <i>Observations</i> level. (These levels are defined in the <i>HE5_PTdeflevel</i> routine.) |

```
status = HE5_PTfwdlinkinfo(pointID2, 1, linkfield);
```

The *linkfield* will contain the string: *ID*.

FORTRAN integer function he5_ptfwdlinkinfo(*pointid,level,linkfield*)

 integer *pointid,status*

 character *(*) *linkfield*

 integer *level*

The equivalent *FORTRAN* code for the example above is:

```
level = 1
```

```
status = he5_ptfwdlinkinfo(pointid, level, linkfield)
```

Return Level Name

HE5_PTgetlevelname

```
herr_t HE5_PTgetlevelname(hid_t pointID,int level, char *levelname, long *strbufsize)
```

| | |
|-------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Point level (0-based) |
| <i>levelname</i> | OUT: Level name |
| <i>strbufsize</i> | OUT: String length of level name |
| Purpose | Returns the name of a level given the level number. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns the name of a level given the level number (0-based). If the user passes NULL for the level name, the routine will return just the string length of the level name (not counting the null terminator). |
| Example | In this example, we return the level name of the 0 th level of the second point defined in the <i>HE5_PTdeflevel</i> section: |

```
status = HE5_PTgetlevelname(pointID2, 0, levelname,  
&strbufsize);
```

The *levelname* will contain the string: *Desc-Loc* and the *strbufsize* variable will be set to 8.

FORTRAN

```
integer function he5_ptgetlevelname(pointid,level,levelname,strbufsz)  
integer      pointid,status,level  
character *(*) levelname  
integer*4     strbufsz
```

The equivalent *FORTRAN* code for the example above is:

```
level = 0  
  
status = he5_ptgetlevelname(pointid, level, levelname,  
strbufsz)
```

Return Record Numbers Related to Level

HE5_PTgetrecnums

herr_t HE5_PTgetrecnums(hid_t *pointID*, int *inlevel*, int *outlevel*, hsize_t *inNrec*,
hssize_t *inRecs*[], hsize_t **outNrec*, hssize_t *outRecs*[])

| | |
|-----------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>inlevel</i> | IN: Level number of input records(0-based) |
| <i>outlevel</i> | IN: Level number of output records(0-based) |
| <i>inNrec</i> | IN: Number of records in the <i>inRecs</i> array |
| <i>inRecs</i> | IN: Array containing the input record numbers. |
| <i>outNrec</i> | OUT: Number of records in the <i>outRecs</i> array |
| <i>outRecs</i> | OUT Array containing the output record numbers. |
| Purpose | Returns the record numbers in one level corresponding to a group of records in a different level. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | The records in one level are related to those in another through the link field. These in turn are related to the next. In this way each record in any level is related to others in all the levels of the point structure. The purpose of <i>HE5_PTgetrecnums</i> is to return the record numbers in one level that are connected to a given set of records in a different level. Note that the two levels need not be adjacent. |
| Example | In this example, we get the record number in the second level that are related to the first record in the first level. |

```
nrec = 1;  
recs[0] = 0;  
inLevel = 0;  
outLevel = 1;  
status = HE5_PTgetrecnums(pointID2, inLevel, outLevel, nrec,  
recs, &outNrec, outRecs);
```

FORTRAN Not available with this release.

Return Information About Group Attribute

HE5_PTgrpattrinfo

```
herr_t HE5_PTgrpattrinfo(hid_t pointID, const char *attrname, H5T_class_t *  
    numbertype, hsize_t *count)
```

| | |
|-------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: Attribute name |
| <i>numbertype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of total bytes in attribute |
| Purpose | Returns information about attribute associated with the point “Data” group |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of an attribute associated with the point “Data” group. |
| Example | In this example, we return information about the <i>GroupFloat</i> attribute. |

```
status = HE5_PTgrpattrinfo(pointID,  
    "GroupFloat", &nt, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 4.

FORTRAN integer function he5_ptgrpattrinfo(*pointid*,*attrname*,*ntype*,*count*)
 integer *pointid*,*status*
 integer*4 *count*
 integer *ntype*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptgrpattrinfo(pointid, "GroupFloat", ntype,  
    count)
```

Retrieve Information About Point Attributes

HE5_PTinqattrs

long HE5_PTinqattrs(hid_t *pointID*, char **attrlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrlist</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about attributes defined in point. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the attributes defined in a point structure. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = HE5_PTinqattrs(pointID, NULL, strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = HE5_PTinqattrs(pointID, attrlist, strbufsize);
```

The variable, *attrlist*, will be set to:

"*attrOne,attr_2*".

FORTRAN integer*4 function he5_ptinqattrs(*pointid,attrlist,strbufsz*)
 integer *pointid*
 character *(*) *attrlist*
 integer*4 *nattr, strbufsz*

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_ptinqattrs(pointid, attrlist, strbufsz)
```

Retrieve Information About Group Attributes

HE5_PTinqgrpattrs

long HE5_PTinqgrpattrs(hid_t *pointID*, char **attrlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrlist</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about attributes defined in point “Data” group. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the attributes defined in the “Data” group of point structure. We assume that there are two attributes stored, *GrpAttrOne* and *GrpAttr_2*:

```
nattr = HE5_PTinqgrpattrs(pointID, NULL, strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 20.

```
nattr = HE5_PTinqgrpattrs(pointID, attrlist, strbufsize);
```

The variable, *attrlist*, will be set to:

"GrpAttrOne,GrpAttr_2".

FORTRAN integer*4 function he5_ptinqgrpattrs(*pointid,attrlist,strbufsz*)
 integer *pointid*
 character *(*) *attrlist*
 integer*4 *nattr,strbufsz*

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_ptinqgrpattrs(pointid, attrlist, strbufsz)
```

Retrieve Information About Level Attributes

HE5_PTinqlocattrs

```
long HE5_PTinqlocattrs(hid_t pointID, const char *levelname, char *attrlist, long  
*strbufsize)
```

| | |
|-------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>levelname</i> | IN: Level name |
| <i>attrlist</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about attributes defined for a specified level in a point. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |
| Example | In this example, we retrieve information about the local attributes defined for the <i>Observations</i> level in a point structure. We assume that there are two attributes stored, <i>LocAttrOne</i> and <i>LocAttrTwo</i> : |

```
nattr = HE5_PTinqlocattrs(pointID, "Observations", NULL,  
strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 21.

```
nattr = HE5_PTinqlocattrs(pointID, levelname, attrlist,  
strbufsize);
```

The variable, *attrlist*, will be set to:

"*LocAttrOne*,*LocAttrTwo*".

FORTRAN

```
integer*4      function  
he5_ptinqlocattrs(pointid,levelname,attrlist,strbufsz)  
integer        pointid
```

```
character *(*) levelname, attrlist  
integer*4      nattr,strbufsz
```

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_ptinqlocattrs(pointid, levelname, attrlist,  
strbufsz)
```

Retrieve Point Structures Defined in HDF-EOS File

HE5_PTinqpoint

int HE5_PTinqpoint(const char * *filename*, char **pointlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>filename</i> | IN: HDF-EOS filename |
| <i>pointlist</i> | OUT: Point list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of point list |
| Purpose | Retrieves number and names of points defined in HDF-EOS file. |
| Return value | Number of points found if successful or FAIL (-1) otherwise. |
| Description | The point list is returned as a string with each point name separated by commas. If <i>pointlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . If <i>strbufsize</i> is also set to NULL, the routine returns just the number of points. Note that <i>strbufsize</i> does not count the null string terminator. |
| Example | In this example, we retrieve information about the points defined in an HDF-EOS file, <i>Point.he5</i> . We assume that there are two points stored, <i>PointOne</i> and <i>Point_2</i> : |
| | <pre>npoint = HE5_PTinqpoint("Point.he5", NULL, strbufsize);</pre> |
| | The parameter, <i>npoint</i> , will have the value 2 and <i>strbufsize</i> will have value 16. |
| | <pre>npoint = HE5_PTinqpoint("Point.he5", pointlist, strbufsize);</pre> |
| | The variable, <i>pointlist</i> , will be set to: “ <i>PointOne,Point_2</i> ”. |

FORTRAN integer function he5_ptinqpoint(*filename*,*pointlist*,*strbufsz*)
integer *npoint*
character *(*) *pointlist*
integer*4 *strbufsz*

The equivalent *FORTRAN* code for the example above is:

```
npoint = he5_ptinqpoint("Point.he5", pointlist, strbufsz)
```

Return Index Number of a Named Level

HE5_PTlevelindx

```
int HE5_PTlevelindx(hid_t pointID, const char *levelname)
```

pointID IN: Point ID returned by HE5_PTcreate or HE5_PTattach

levelname IN: Level Name

Purpose Returns the level index (0-based) for a given (named) level.

Return value Returns the level index if successful or FAIL (-1) otherwise.

Description This routine returns the level index for a give level specified by name.

Example In this example, we return the level index of the *Observations* level in the multilevel point structure defined in *HE5_PTdeflevel*.

```
levindx = HE5_PTlevelindx(pointID2, "Observations");
```

The *levindx* variable will have the value 1.

FORTRAN integer function he5_ptlevelindx(*pointid,levelname*)

integer *pointid,levindx*

character *(*) *levelname*

The equivalent *FORTRAN* code for the example above is:

```
levindx = he5_ptlevelindx(pointid, "Observations")
```

Return Information on Fields in a Given Level

HE5_PTlevelinfo

herr_t HE5_PTlevelinfo(hid_t *pointID*, int *level*, HE5_CmpDTSinfo **info*)

| | |
|----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Point level (0-based) |
| <i>info</i> | OUT: C-data structure containing the level information |
| Purpose | Returns information on fields in a given level. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or level number. |
| Description | This routine returns information about the fields in a given level. |
| Example | In this example we return information about the <i>Desc-Loc</i> (1st) level defined previously. <pre>HE5_CmpDTSinfo lev0_info; status = HE5_PTlevelinfo(pointID2, 0, &lev0_info);</pre> The <i>lev0_info.nfields</i> data member will be set to 5. The <i>lev0_info.fieldname</i> array will be "Time,Longitude,Latitude,Channel,Value". |
| FORTRAN | See Example 4 from Section 7.1.1.2 of Volume 1 (Overview of Examples). |

Return Information About Level Attribute

HE5_PTlocattrinfo

```
herr_t HE5_PTlocattrinfo(hid_t pointID, const char *levelname, const char  
                         *attrname, H5T_class_t *numbertype, hsize_t *count)
```

| | |
|-------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_Ptattach |
| <i>levelname</i> | IN: Level name |
| <i>attrname</i> | IN: Attribute name |
| <i>numbertype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of total bytes in attribute |
| Purpose | Returns information about point level attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of an attribute associated with a specified level. |
| Example | In this example, we return information about the <i>LocalFloat</i> attribute associated with the level <i>Observations</i> . |

```
status = HE5_PTattrinfo(pointID,  
                        "Observations", "LocalFloat", &nt, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 4.

FORTRAN

```
integer function he5_ptlocattrinfo(pointid,levelname,attrname,ntype,count)  
integer      pointid,status,ntype  
character *(*) levelname, attrname  
integer*4      count
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptlocattrinfo(pointid, "Observations",  
                           "LocalFloat", ntype, count)
```

Return Number of Fields Defined in a Level

HE5_PTnfields

int HE5_PTnfields(hid_t *pointID*, int *level*, char **fieldlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Level number (0-based) |
| <i>fieldlist</i> | OUT: Field list (entries separated by commas) |
| <i>strbufsize</i> | OUT: Size in bytes of fieldlist for level |
| Purpose | Returns number of fields in a level and the size of the fieldlist. |
| Return value | Returns number of fields if successful or FAIL (-1) otherwise. |
| Description | This routine returns the number of fields in a level and the size of the comma-separated fieldlist. This value does NOT count the null character at the end of the string. |
| Example | In this example we retrieve the number of levels in the 2nd point defined previously: |

```
nflds = HE5_PTnfields(pointID2, 0, NULL, &strbufsize);
nflds = HE5_PTnfields(pointID2, 0, fieldlist, &strbufsize);
```

The *nflds* variable will be 5 and the *strbufsize* variable equal to 38.

FORTRAN integer function he5_ptnfields(*pointid2,level,fieldlist,strbufsz*)
 integer *pointid2,level,nflds*
 character *(*) *fieldlist*
 integer*4 *strbufsz*

The equivalent *FORTRAN* code for the example above is:

```
level = 0
nflds = he5_ptnfields(pointid2, level, fieldlist, strbufsz)
```

Return Number of Levels in a Point Structure

HE5_PTnlevels

int HE5_PTnlevels(hid_t *pointID*)

| | |
|----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| Purpose | Returns number of levels in a point. |
| Return value | Returns number of levels if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID. |
| Description | This routine returns the number of levels in a point. |
| Example | In this example we retrieve the number of levels in the 2nd point defined previously: |

```
nlevels = HE5_PTnlevels(pointID2);
```

The *nlevels* variable will be 2.

FORTRAN integer function he5_ptnlevels(*pointid*)

```
              integer      pointid,nlevels
```

The equivalent *FORTRAN* code for the example above is:

```
nlevels = he5_ptnlevels(pointid)
```

Return Number of Records in a Given Level

HE5_PTnrecs

hsize_t HE5_PTnrecs(hid_t *pointID*, int *level*)

| | |
|----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Level number (0-based) |
| Purpose | Returns number of records in a given level. |
| Return value | Returns number of records in a given level if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point id or level number. |
| Description | This routine returns the number of records in a given level. |
| Example | In this example we retrieve the number of records in the first level of the 2nd point defined previously: |

```
nrecs = HE5_PTnrecs(pointID2, 0);
```

FORTRAN integer function he5_ptnrecs(*pointid,level*)
 integer *pointid2,level*

The equivalent *FORTRAN* code for the example above is:

```
level = 0  
status = he5_ptnrecs(pointid2, level)
```

Open HDF-EOS File

HE5_POpen

hid_t HE5_POpen(const char *filename, uintn access)

| | |
|-----------------|--|
| <i>filename</i> | IN: Complete path and filename for the file to be opened |
| <i>access</i> | IN: H5F_ACC_RDONLY, H5F_ACC_RDWR or H5F_ACC_TRUNC |
| Purpose | Opens or creates HDF-EOS file in order to create, read, or write a point. |
| Return value | Returns the point file ID (fid) if successful or FAIL (-1) otherwise. |
| Description | This routine creates a new file or opens an existing one, depending on the access parameter. |

Access codes:

| | |
|----------------|--|
| H5F_ACC_RDONLY | Open for read only. If file does not exist, error |
| H5F_ACC_RDWR | Open for read/write. If file does not exist, error |
| H5F_ACC_TRUNC | If file exists, delete it, then open a new file for read/write |

| | |
|---------|--|
| Example | In this example, we create a new point file named, <i>Point.he5</i> . It returns the file handle, <i>fid</i> . |
|---------|--|

```
fid = HE5_POpen( "Point.he5" , H5F_ACC_TRUNC );
```

FORTRAN integer function he5_ptopen(*filename,flag*)
 integer *fid*
 character *(*) *filename*
 integer *HE5_HDFE_TRUNC*
 parameter (*HE5_HDFE_TRUNC*=2)

The equivalent **FORTRAN** code for the example above is:

```
fid = he5_ptopen( "Point.he5" , HE5_HDFE_TRUNC )
```

Read Point Attribute

HE5_PTreadattr

herr_t HE5_PTreadattr(hid_t *pointID*, const char **attrname*, void **datbuf*)

| | |
|-----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | IN: Buffer allocated to hold attribute values |
| Purpose | Reads attribute from a point. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read floating point attribute with the name "ScalarFloat": |

```
status = HE5_PTreadattr(pointID, "ScalarFloat", &attr_val);
```

FORTRAN integer function he5_ptreadatt(*pointid,attrname,buffer*)
 integer *pointid,status*
 character *(*) *attrname*
 <valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptreadatt(pointid, "ScalarFloat", buffer)
```

Read Point Group Attribute

HE5_PTreadgrpattr

herr_t HE5_PTreadgrpattr(hid_t *pointID*, const char **attrname*, void **datbuf*)

| | |
|-----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | IN: Buffer allocated to hold attribute values |
| Purpose | Reads attribute associated with the “Data” group in a point. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read floating point attribute with the name "GroupFloat": |

```
status = HE5_PTreadgrpattr(pointID, "GroupFloat",  
&attr_val);
```

FORTRAN integer function he5_ptreadgrpattr(*pointid,attrname,buffer*)
 integer *pointid,status*
 character *(*) *attrname*
 <valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptreadgrpattr(pointid, "GroupFloat", buffer)
```

Read Point Level Attribute

HE5_PTreadlocattr

```
herr_t HE5_PTreadlocattr(hid_t pointID, const char *levelname, const char  
*attrname, void *datbuf)
```

| | |
|------------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>levelname</i> | IN: Level name |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | IN: Buffer allocated to hold attribute values |
| Purpose | Reads attribute associated with a specified level in a point. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read floating point attribute with the name "LocalFloat" defined in the <i>Observations</i> level: |

```
status = HE5_PTreadlocattr(pointID, "Observations",  
"LocalFloat", &attr_val);
```

FORTRAN integer function he5_ptreadlocattr(*pointid,levelname,attrname,buffer*)
 integer *pointid,status*
 character *(*) *levelname,attrname*
 <valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptreadlocattr(pointid,  
"Observations", "LocalFloat", buffer)
```

Read Records From a Point Level

HE5_PTreadlevel

```
herr_t HE5_PTreadlevel(hid_t pointID, int level, HE5_CmpDTSinfo *inStruct,  
size_t *size, void *buffer)
```

| | |
|-----------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Level to read (0-based) |
| <i>inStruct</i> | IN: C-data structure containing information about specified level. |
| <i>size_t</i> | IN: Size (in bytes) of data structure to read data to. |
| <i>buffer</i> | OUT: Buffer to store data |
| Purpose | Reads data from a point level. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or level number. |
| Description | This routine reads data from the specified fields and records of a single level in a point. An appropriate read buffer must be defined by the user. |
| Example | In this example we read records from the first level in the point referred to by the point ID, <i>pointID1</i> . User should define data structure to store the output data, first. Suppose the user defined data structure to read the output data to is “Sensor”. |

```
CmpDTSinfo    lev0_info;  
CmpDTSinfo    input_info;  
Sensor        *buffer;  
  
/* Get all necessary information about level first */  
status = HE5_Ptlevelinfo(pointID1, 0, &lev0_info);  
/* Set up input data structure and calculate the data size */  
nrecs = HE5_Ptnrecs(pointID1, 0);  
buffer = (Sensor *)calloc(nrecs, sizeof(Sensor));  
status = HE5_PTreadlevel(pointID1, 0, &lev0_info, datasize,  
buffer);
```

| | |
|---------|--|
| FORTRAN | See Example 4 from Section 7.1.1.2 of Volume 1 (Overview of Examples). |
|---------|--|

Return Data Type Information for a Level

HE5_PTinqdatatype

```
herr_t HE5_PTinqdatatype(hid_t pointID, const char *levelname, const char  
                         *attrname, int fieldgroup, hid_t *datatype, H5T_class_t  
                         *classid, H5T_order_t *order, size_t *size)
```

| | | |
|-------------------|--|--|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>levelname</i> | IN: | Level name |
| <i>attrname</i> | IN: | Attribute name |
| <i>fieldgroup</i> | IN: | Field group flag: HE5_HDFE_DATAGROUP - 1 HE5_HDFE_ATTRGROUP - 2 HE5_HDFE_GRPATTRGROUP - 3 HE5_HDFE_LOCATTRGROUP - 4 |
| <i>datatype</i> | OUT: | Data type ID |
| <i>classID</i> | OUT: | Data type class ID |
| <i>order</i> | OUT: | Data type byte order |
| <i>size</i> | OUT: | Data type size (in bytes) |
| Purpose | Returns data type information about specified level in point. | |
| Return value | Returns size in bytes of specified fields if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or level name. | |
| Description | This routine returns information about level data in a point. | |
| Example | In this example we return the data type information for the <i>Observations</i> level in the point defined in the <i>HE5_PTdeflevel</i> routine. <pre>status = HE5_PTinqdatatype(pointID, "Observations", NULL, fieldgroup, &datatype, &classid, &order, &size);</pre> | |

FORTRAN integer function
 he5_ptinqdatatype(*pointid*,*levelname*,*attrname*,*fldgrp*,*dtype*,*classid*,*order*,
 size)
 integer *pointid*,*status*

```
integer      dtype, classid, order
integer*4     size
character *(*) levelname
integer       HE5_HDFE_DATAGROUP
parameter     (HE5_HDFE_DATAGROUP=1)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ptinqdatatype(pointid1, "Observations", "",  
    dtype, classid, order, size)
```

Update Records in a Point Structure

HE5_PTupdatelevel

herr_t HE5_PTupdatelevel(hid_t *pointID*, int *level*, char* *fieldlist*, hsize_t *nrec*,
hssize_t *recls[]*, void **data*)

| | |
|------------------|---|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Level to update (0-based) |
| <i>fieldlist</i> | IN: List of fields to update |
| <i>nrec</i> | IN: Number of records to update |
| <i>recls</i> | IN: Record number of records to update (0 - based) |
| <i>data</i> | IN: Data buffer to be written |
| Purpose | Updates (corrects) data to a point level. |
| NOTE: | Currently updating of a whole record is supported. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or unknown fieldname. |
| Description | This routine updates the specified fields and records of a single level. |
| Example | In this example we update records 0, 2, and 3 for the field <i>Concentration</i> in first level in the point referred to by the point ID, <i>pointID1</i> . |
| | <pre>hsize_t recls[3] = {0, 2, 3}; /* Fill Data Buffer */ status = HE5_PTupdatelevel(pointID1, 0, "Concentration", 3, recls, datbuf);</pre> |
| | The user may update a single record or all records . |
| FORTRAN | See Example 5 from Section 7.1.1.2 of Volume 1 (Overview of Examples). |

Write/Update Point Attribute

HE5_PTwriteattr

```
herr_HE5_PTwriteattr(hid_t pointID, const char *attrname, int ntype, hsize_t count, void *datbuf)
```

| | | |
|-----------------|---|---|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: | Attribute name |
| <i>ntype</i> | IN: | Number type of attribute |
| <i>count</i> | IN: | Number of values to store in attribute |
| <i>datbuf</i> | IN: | Attribute values |
| Purpose | Writes/Updates attribute in a point. | |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type. | |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. | |
| Example | In this example, we write a floating point attribute with the name "ScalarFloat" and the value 3.14: | |

```
attr_val = 3.14;  
  
status = HE5_PTwriteattr(pointid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_PTwriteattr(pointid, "ScalarFloat",  
H5T_NATIVE_FLOAT, 1, &attr_val);
```

FORTRAN integer function he5_ptwriteattr(*pointid,attrname,ntype,count,buffer*)
 integer *pointid,status,ntype*

```
character *(*) attrname
integer*4      count
<valid type>  buffer(*)
integer        HE5_HDFE_NATIVE_FLOAT
parameter      (HE5_HDFE_NATIVE_FLOAT=1)
```

The equivalent *FORTRAN* code for the example above is:

```
count = 1
status = he5_ptwriteattr(pointid, "ScalarFloat",
HE5_HDFE_NATIVE_FLOAT, count, buffer)
```

Write/Update Point Group Attribute

HE5_PTwritegrpattr

herr_HE5_PTwritegrpattr(hid_t *pointID*, const char **attrname*, int *ntype*, hsize_t *count*, void **datbuf*)

| | | |
|-----------------|-----|---|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>attrname</i> | IN: | Attribute name |
| <i>ntype</i> | IN: | Number type of attribute |
| <i>count</i> | IN: | Number of values to store in attribute |
| <i>datbuf</i> | IN: | Attribute values |
| Purpose | | Writes/Updates attribute associated with the “Data” group in a point. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type. |
| Description | | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. |
| Example | | In this example, we write a floating point group attribute with the name "GroupFloat" and the value 3.14: |

```
attr_val = 3.14;

status = HE5_PTwritegrpattr(pointid, "GroupFloat",
H5T_NATIVE_FLOAT, 1, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;

status = HE5_PTwritegrpattr(pointid, "GroupFloat",
H5T_NATIVE_FLOAT, 1, &attr_val);
```

FORTRAN integer function he5_ptwritegrpattr(*pointid,attrname,ntype,count,buffer*)
 integer *pointid,status,ntype*

```
character *(*) attrname
integer*4      count
<valid type>   buffer(*)
integer        HE5_HDFE_NATIVE_FLOAT
parameter      (HE5_HDFE_NATIVE_FLOAT=1)
```

The equivalent *FORTRAN* code for the example above is:

```
count = 1
status = he5_ptwritegrpattr(pointid, "GroupFloat",
    HE5_HDFE_NATIVE_FLOAT, count, buffer)
```

Write New Records to a Point Level

HE5_PTwritelevel

herr_t HE5_PTwritelevel(hid_t *pointID*, int *level*, hsize_t *nrec*[], size_t **size*, void **data*)

| | |
|----------------|--|
| <i>pointID</i> | IN: Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>level</i> | IN: Level to write (0-based) |
| <i>nrec</i> | IN: Number of records to write |
| <i>size</i> | IN: Data size (bytes) to write |
| <i>data</i> | IN: Data buffer to be written to the level |
| Purpose | Writes (appends) new records to a point level. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or level number. |
| Description | This routine writes (appends) full records to a level. The data buffer should be represented by the array of C-data type structures. The structure type should be consistent with that used in <i>HE5_PTdeflevel()</i> . |
| Example | In this example we write 5 records to the first level in the point referred to by the point ID, <i>pointID1</i> . |
| | <pre>/* Fill Data Buffer */ /* Calculate the data size (bytes) */ status = HE5_PTwritelevel(pointID1, 0, 5, datasize, datbuf);</pre> |
| FORTRAN | See Example 3 from Section 7.1.1.2 of Volume 1 (Overview of Examples). |

Write/Update Point Level Attribute

HE5_PTwritelocattr

herr_HE5_PTwritelocattr(hid_t *pointID*, const char **levelname*, const char **attrname*, int *ntype*, hsize_t *count*, void **datbuf*)

| | | |
|------------------|-----|---|
| <i>pointID</i> | IN: | Point ID returned by HE5_PTcreate or HE5_PTattach |
| <i>levelname</i> | IN: | Level name |
| <i>attrname</i> | IN: | Attribute name |
| <i>ntype</i> | IN: | Number type of attribute |
| <i>count</i> | IN: | Number of values to store in attribute |
| <i>datbuf</i> | IN: | Attribute values |
| Purpose | | Writes/Updates attribute associated with a specified level in a point. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper point ID or number type. |
| Description | | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. |

Example In this example, we write a floating point attribute with the name "LocalFloat" and the value 3.14 associated with the level "Observations":

```
attr_val = 3.14;  
  
status = HE5_PTwritelocattr(pointid, "Observations",  
"LocalFloat", H5T_NATIVE_FLOAT, 1, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_PTwritelocattr(pointid, "Observations",  
"LocalFloat", H5T_NATIVE_FLOAT, 1, &attr_val);
```

FORTRAN integer function
he5_ptwritelocattr(*pointid,levelname,attrname,ntype,count,buffer*)

```
integer      pointid,status,ntype
character *(*) attrname,levelname
integer*4     count
<valid type> buffer(*)
integer       HE5_HDFE_NATIVE_FLOAT
parameter     (HE5_HDFE_NATIVE_FLOAT=1)
```

The equivalent *FORTRAN* code for the example above is:

```
count = 1
status = he5_ptwritelocattr(pointid, "Observations",
"LocalFloat", HE5_HDFE_NATIVE_FLOAT, count, buffer)
```

2.1.2 Swath Interface Functions

This section contains an alphabetical listing of all the functions in the Swath interface. The functions are alphabetized based on their C-language names.

Return Information About an Alias

HE5_SWaliasinfo

```
herr_t HE5_SWaliasinfo(hid_t swathID, int fldgroup, const char *aliasname, int  
*length, char *buffer)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fldgroup</i> | IN: Field group flag |
| <i>aliasname</i> | IN: Name of alias to retrieve information about |
| <i>length</i> | OUT: Size of buffer in bytes |
| <i>buffer</i> | OUT: Buffer with original field name |
| Purpose | Return information about an alias |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | Creates aliases that can be used to refer to a Swath data field in addition to the name of the field. |
| Example | In this example, we create and alias for the data field <i>Temperature</i> . |

```
status = HE5_SWaliasinfo(swathID, HE5_HDFE_DATAGROUP,  
aliasname, length, NULL);  
  
namebuffer = (char *)calloc(length + 1, sizeof(char));  
  
status = HE5_SWaliasinfo(swathID, HE5_HDFE_DATAGROUP,  
aliasname, length, namebuffer);
```

FORTRAN integer function he5_swaliasinfo (*swathid*, *fldgroup*, *aliasname*, *length*,
buffer)
 integer *swathid*,*status*
 character(*) *fldgroup*
 character(*) *aliasname*
 integer(*) *length*
 character(*) *buffer*

The equivalent *FORTRAN* code for the first example above is:

```
aliaslist = "temps 0 to 30"  
  
status = he5_swaliasinfo(swathid, "Temperature", aliaslist,  
length, buffer)
```

Attach to an Existing Swath Structure

HE5_SWAttach

hid_t HE5_SWAttach(hid_t *fid*, const char **swathname*)

| | |
|------------------|---|
| <i>fid</i> | IN: Swath file ID returned by HE5_SWopen |
| <i>swathname</i> | IN: Name of swath to be attached |
| Purpose | Attaches to an existing swath within the file. |
| Return value | Returns the swath handle (swathID) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath file id or swath name. |
| Description | This routine attaches to the swath using the <i>swathname</i> parameter as the identifier. |
| Example | In this example, we attach to the previously created swath, "ExampleSwath", within the HDF-EOS file, <i>Swath.he5</i> , referred to by the handle, <i>fid</i> : |

```
swathID = HE5_SWAttach(fid, "ExampleSwath");
```

The swath can then be referenced by subsequent routines using the handle, *swathID*.

FORTRAN integer function he5_swattach(*fid*,*swathname*)

integer *fid*

character*(*) *swathname*

The equivalent *FORTRAN* code for the example above is:

```
swathid = he5_swattach(fid, "ExampleSwath")
```

Return Information About a Swath Attribute

HE5_SWattrinfo

```
herr_t HE5_SWattrinfo(hid_t swathID, const char *attrname, H5T_class_t *ntype,  
                      hsize_t *count)
```

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | OUT: Number type of attribute |
| <i>count</i> | OUT: Number of total bytes in attribute |
| Purpose | Returns information about a swath attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a swath attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_SWattrinfo(swathID, "ScalarFloat", &nt, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 4.

FORTRAN integer function he5_swattrinfo(*swathid*, *attrname*, *ntype*, *count*)
 integer *swathid*
 character(*) *attrname*
 integer *ntype*
 integer*4 *count*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swattrinfo(swathid, "ScalarFloat", ntype, count)
```

Close an HDF-EOS File

HE5_SWclose

herr_t HE5_SWclose(hid_t *fid*)

fid IN: Swath file ID returned by HE5_SWopen

Purpose Closes file.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This routine closes the HDF-EOS Swath file.

Example

```
status = HE5_SWclose(fid);
```

FORTRAN integer function he5_swclose(*fid*)
integer *fid*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swclose(fid)
```

Retreive Compression Information for Field

HE5_SWcompinfo

```
herr_t HE5_SWcompinfo(hid_t swathID, const char *fieldname, int *compcode,  
                      int compparm[])
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Fieldname |
| <i>compcode</i> | OUT: HDF compression code |
| <i>compparm</i> | OUT: Compression parameters |
| Purpose | Retrieves compression information about a field. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. |
| Description | This routine returns the compression code and compression parameters for a given field. |
| Example | To retreive the compression information about the <i>Opacity</i> field defined in the <i>HE5_SWdefcomp</i> function: |

```
status = HE5_SWcompinfo(swathID, "Opacity", &compcode,  
                        compparm);
```

The *compcode* parameter will be set to 4 and *compparm*[0] to 5.

FORTRAN

```
integer function he5_swcompinfo(gridid,fieldname compcode, compparm)  
    integer         swathid  
    character(*)   fieldname  
    integer         compcode  
    integer         compparm(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swcompinfo(swathid, 'Opacity', compcode,  
                        compparm)
```

The *compcode* parameter will be set to 4 and *compparm*(1) to 5.

Create a New Swath Structure

HE5_SWcreate

hid_t HE5_SWcreate(hid_t *fid*, const char **swathname*)

| | |
|------------------|--|
| <i>fid</i> | IN: Swath file ID returned by HE5_SWopen |
| <i>swathname</i> | IN: Name of swath to be created |
| Purpose | Creates a swath within the file. |
| Return value | Returns the swath handle (<i>swathID</i>) if successful or FAIL (-1) otherwise. |
| Description | The swath is created as a Group within the HDF-EOS file with the name <i>swathname</i> . |
| Example | In this example, we create a new swath structure, <i>ExampleSwath</i> , in the previously created file, <i>Swath.he5</i> . |

```
swathID = HE5_SWcreate(fid, "ExampleSwath");
```

The swath structure is referenced by subsequent routines using the handle, *swathID*.

FORTRAN integer function he5_swcreate(*fid*,*swathname*)
 integer *fid*
 character*(*) *swathname*

The equivalent *FORTRAN* code for the example above is:

```
swathid = he5_swcreate(fid, "ExampleSwath")
```

Define a Longitude-Latitude Box Region for a Swath

HE5_SWdefboxregion

```
hid_t HE5_SWdefboxregion(hid_t swathID, double cornerlon[], double  
                           cornerlat[], int mode)
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>cornerlon</i> | IN: Longitude in decimal degrees of box corners |
| <i>cornerlat</i> | IN: Latitude in decimal degrees of box corners |
| <i>mode</i> | IN: Cross Track inclusion mode |
| Purpose | Defines a longitude-latitude box region for a swath. |
| Return value | Returns the swath region ID if successful or FAIL (-1) otherwise. |
| Description | This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the <i>HE5_SWextractregion</i> routine to read all the entries of a data field within the region. A cross track is within a region if 1) its midpoint is within the longitude-latitude "box" (HE5_HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HE5_HDFE_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HE5_HDFE_ANYPOINT), depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the region even though a particular element of the cross track might be outside the region. The swath structure must have both <i>Longitude</i> and <i>Latitude</i> (or <i>Colatitude</i>) fields defined. |
| Example | In this example, we define a region bounded by the 3 degrees longitude, 5 degrees latitude and 7 degrees longitude, 12 degrees latitude. We will consider a cross track to be within the region if its midpoint is within the region. <pre>cornerlon[0] = 3.; cornerlat[0] = 5.; cornerlon[1] = 7.; cornerlat[1] = 12.; regionID = HE5_SWdefboxregion(swathID, cornerlon, cornerlat, HE5_HDFE_MIDPOINT);</pre> |

```
FORTRAN    integer function he5_swdefboxreg(swathid, cornerlon, cornerlat, mode)
            integer      swathid
            real*8       cornerlon(*)
            real*8       cornerlat(*)
            integer      mode
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_MIDPOINT=0)
cornerlon(1) = 3.
cornerlat(1) = 5.
cornerlon(2) = 7.
cornerlat(2) = 12.
regionid = he5_swdefboxreg(swathid, cornerlon, cornerlat,
                           HE5_HDFE_MIDPOINT)
```

Define Chunking Parameters

HE5_SWdefchunk

```
herr_t HE5_SWdefchunk(hid_t swathID, int chunk_rank, const hsize_t  
*chunk_dims)
```

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach

chunk_rank IN: The number of chunk dimensions

chunk_dims IN: Chunk dimensions

Purpose Defines chunking for subsequent field definitions

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description This routine defines the chunking dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using *HE5_SWdefcomp*. The number of tile dimensions and subsequent field dimensions must be the same.

Example We will define chunking for a two-dimensional field of size
2400 x 3600 .

```
chunk_dims[0] = 100;  
chunk_dims[1] = 360;  
  
status = HE5_SWdefchunk(swathID, 2, chunk_dims);
```

FORTRAN integer function he5_swdefchunk(*swathid*, *chunk_rank*,
chunk_dims)
integer *swathid*
integer *chunk_rank*
integer*4 *chunk_dims(*)*

The equivalent *FORTRAN* code for the example above is:

```
chunk_dims(1) = 360  
chunk_dims(2) = 100  
chunk_rank     = 2  
  
status = he5_swdefchunk(swathid, chunk_rank, chunk_dims)
```

Define Compression with Data Chunking

HE5_SWdefcomchunk

herr_t HE5_SWdefcomchunk(hid_t *swathID*, int *compcode*, int **compparm*, int *ndims*, const hsize **dim*)

| | | |
|-----------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>compcode</i> | IN: | Compression method flag |
| <i>compparm</i> | IN: | Array of compression parameters |
| <i>ndims</i> | IN: | Rank of a field to compress |
| <i>dim</i> | IN: | Array of sizes of chunk |
| Purpose | | Compress the data field |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | This function allows the user to set compression for a data field with automatic chunking |
| Example | | In this example, we set (DEFLATE) compression for a field that is defined right after this call |

```
ndims      = 2
compcode   = 4;
compparm[0] = 6;
dim[0]     = 100;
dim[1]     = 200;
status = HE5_SWdefcomchunk(swathID, compcode, compparm,
                           ndims, dim);
```

FORTRAN integer function he5_swdefcomch(swathid,compcode,compparm,
 ndims, dim)

```
integer      swathid
integer      compcode
integer      compparm(*)
integer      ndims
integer*4    dim(*)
```

The equivalent *FORTRAN* code for the example above is:

```
compcode      = 4
compparm(1)   = 6
ndims         = 2
dim(1)        = 200
dim(2)        = 100
status = he5_swdefcomch(swathid, compcode, compparm, ndims,
dim)
```

Set Swath Field Compression

HE5_SWdefcomp

herr_t HE5_SWdefcomp(hid_t *swathID*, int *compcode*, int **compparm*)

| | |
|--|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>compcode</i> | IN: HDF compression code |
| <i>compparm</i> | IN: Compression parameters (if applicable) |
| Note: Only deflate compression is available in this release. | |
| Purpose | Sets the field compression for all subsequent field definitions. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. |
| Description | This routine sets the HDF field compression for subsequent swath field definitions. The compression does not apply to one-dimensional fields. The compression schemes currently supported is: deflate (gzip) (HE5_HDFE_COMP_DEFLATE=4) and no compression (HE5_HDFE_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard <i>HE5_SWwritefield</i> routine, however, the entire field must be written in a single call. Any portion of a compressed field can then be accessed with the <i>HE5_SWreadfield</i> routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters. |

| | |
|---------|---|
| Example | Suppose we wish to compress the <i>Pressure</i> using run length encoding, the <i>Opacity</i> field using deflate compression, the <i>Spectra</i> field with skipping Huffman compression, and use no compression for the <i>Temperature</i> field. |
|---------|---|

```
status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_DEFLATE,
NULL);

status = HE5_SWdefdatafield(swathID, "Pressure",
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);

compparm[0] = 5;

status = HE5_SWdefdatafield(swathID, "Opacity",
"Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);
```

```

status = HE5_SWdefdatafield(swathID, "Spectra",
    "Bands,Track,Xtrack", NULL, H5T_NATIVE_FLOAT, HDFE_NOMERGE);

status = HE5_SWdefcomp(swathID, HE5_HDFE_COMP_NONE, NULL);

status = HE5_SWdefdatafield(swathID, "Temperature",
    "Track,Xtrack", NULL, H5T_NATIVE_FLOAT, 0);

```

Note that the HE5_HDFE_AUTOMERGE/MERGE parameter is ignored in the *Temperature* field definition.

FORTRAN integer function he5_swdefcomp(*swathid*, *compcode*, *compparm*)
 integer *swathid*
 integer *compcode*
 integer *compparm*(*)

The equivalent *FORTRAN* code for the example above is:

```

parameter (HE5_HDFE_NATIVE_FLOAT=1)

parameter (HE5_HDFE_COMP_NONE=0)

parameter (HE5_HDFE_COMP_DEFLATE=4)

integer compparm(5)

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_DEFLATE,
compparm);

status = he5_swdefdfld(swathid, "Pressure", "Xtrack,Track",
" ", HE5_HDFE_NATIVE_FLOAT, 0);

compparm(1) = 5

status = he5_swdefdfld(swathid, "Opacity", "Xtrack,Track",
" ", HE5_HDFE_NATIVE_FLOAT, 0);

status = he5_swdefdfld(swathid, "Spectra", "Xtrack,
Track,Bands", " ", HE5_HDFE_NATIVE_FLOAT, 0)

status = he5_swdefcomp(swathid, HE5_HDFE_COMP_NONE,
compparm)

status = he5_swdefdfld(swathid, "Temperature",
"Xtrack,Track", " ", HE5_HDFE_NATIVE_FLOAT, 0)

```

Define a New Data Field within a Swath

HE5_SWdefdatafield

```
herr_t HE5_SWdefdatafield(hid_t swathID, const char *fieldname, char *dimlist,  
                           char *maxdimlist, hid_t ntype, int merge)
```

| | | |
|-------------------|-----|--|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: | Name of field to be defined |
| <i>dimlist</i> | IN: | The list of data dimensions defining the field |
| <i>maxdimlist</i> | IN: | The list of maximum data dimensions defining the field |
| <i>ntype</i> | IN: | The number type of the data stored in the field |
| <i>merge</i> | IN: | Merge code (HE5_HDFE_NOMERGE (0) - no merge, HE5_HDFE_AUTOMERGE (1) -merge) |

Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”

| | |
|--------------|--|
| Purpose | Defines a new data field within the swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list. |
| Description | This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. |
| Example | In this example, we define a three dimensional data field named <i>Spectra</i> with dimensions <i>Bands</i> , <i>DataTrack</i> , and <i>DataXtrack</i> : |
| | <pre>status = HE5_SWdefdatafield(swathID, "Spectra", "Bands,DataTrack,DataXtrack", " ", H5T_NATIVE_FLOAT, 0);</pre> |
| | Note: To assure that the fields defined by <i>HE5_SWdefdatafield</i> are properly established in the file, the swath should be detached (and then reattached) before writing to any fields. |
| FORTRAN | integer function he5_swdeffld(<i>swathid</i> , <i>fieldname</i> , <i>dimlist</i> , <i>maxdimlist</i> , <i>ntype</i> , <i>merge</i>) |

```
integer      swathid
character(*) fieldname
character(*) dimlist
character(*) maxdimlist
integer      ntype
integer      merge
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_AUTOMERGE=1)
status = he5_swdefdfld(swathid, "Spectra", "DataXtrack",
DataTrack, Bands", " ", HE5_HDFE_NATIVE_FLOAT, 0)
```

Define a New Dimension within a Swath

HE5_SWdefdim

herr_t HE5_SWdefdim(hid_t *swathID*, char **dimname*, hsize_t *dim*)

| | | |
|--|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>dimname</i> | IN: | Name of dimension to be defined |
| <i>dim</i> | IN: | The size of the dimension |
| Note: There are three illegal characters for dimension names: “/”, “;”, “,” | | |

Purpose Defines a new dimension within the swath.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is an improper swath ID.

Description This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.

Example In this example, we define a track geolocation dimension, *GeoTrack*, of size 2000, a cross track dimension, *GeoXtrack*, of size 1000 and two corresponding data dimensions with twice the resolution of the geolocation dimensions:

```
status = HE5_SWdefdim(swathID, "GeoTrack", 2000);
status = HE5_SWdefdim(swathID, "GeoXtrack", 1000);
status = HE5_SWdefdim(swathID, "DataTrack", 4000);
status = HE5_SWdefdim(swathID, "DataXtrack", 2000);
status = HE5_SWdefdim(swathID, "Bands", 5);
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to -1 or equivalently, *H5S_UNLIMITED*:

```
status = HE5_SWdefdim(swathID, "Unlim", H5S_UNLIMITED);
FORTRAN integer function he5_swdefdim(swathid,dimname,dim)
integer         swathid
character(*)   dimname
integer*4       dim
```

The equivalent *FORTRAN* code for the first example above is:

```
dim      = 2000
status = he5_swdefdim(swathid, "GeoTrack", dim)
```

The equivalent *FORTRAN* code for the unlimited dimension example above is:

```
parameter (H5S_UNLIMITED=-1)

status = he5_swdefdim(swathid, "Unlim", H5S_UNLIMITED)
```

Define Mapping between Geolocation and Data Dimensions

HE5_SWdefdimmap

herr_t HE5_SWdefdimmap(hid_t *swathID*, char **geodim*, char **datadim*, hsize_t *offset*, hsize_t *increment*)

| | | |
|------------------|--|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>geodim</i> | IN: | Geolocation dimension name |
| <i>datadim</i> | IN: | Data dimension name |
| <i>offset</i> | IN: | The offset of the geolocation dimension with respect to the data dimension |
| <i>increment</i> | IN: | The increment of the geolocation dimension with respect to the data dimension |
| Purpose | Defines monotonic mapping between the geolocation and data dimensions. | |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is incorrect geolocation or data dimension name. | |
| Description | Typically the geolocation and data dimensions are of different size (resolution). This routine established the relation between the two where the offset gives the index of the data element (0-based) corresponding to the first geolocation element and the increment gives the number of data elements to skip for each geolocation element. If the geolocation dimension begins "before" the data dimension, then the offset is negative. Similarly, if the geolocation dimension has higher resolution than the data dimension, then the increment is negative. | |
| Example | In this example, we establish that (1) the first element of the <i>GeoTrack</i> dimension corresponds to the first element of the <i>DataTrack</i> dimension and the data dimension has twice the resolution as the geolocation dimension, and (2) the first element of the <i>GeoXtrack</i> dimension corresponds to the second element of the <i>DataXtrack</i> dimension and the data dimension has twice the resolution as the geolocation dimension: | |

```
status = HE5_SWdefdimmap(swathID, "GeoTrack", "DataTrack",
0, 2);

status = HE5_SWdefdimmap(swathID, "GeoXtrack", "DataXtrack",
1, 2);
```

```
FORTRAN    integer function he5_swdefmap(swathid,geodim,datadim,offset,increment)
            integer      swathid
            character(*) geodim
            character(*) datadim
            integer*4     offset
            integer*4     increment
```

The equivalent *FORTRAN* code for the second example above is:

```
offset      = 0
increment  = 2

status = he5_swdefmap(swathid, "GeoTrack", "DataTrack",
                      offset, increment)

offset      = 1
increment  = 2

status=he5_swdefmap(swathid, "GeoXtrack", "DataXtrack", offset,
                     increment)
```

Define a new Geolocation Field within a Swath

HE5_SWdefgeofield

```
herr_t HE5_SWdefgeofield(hid_t swathID, const char *fieldname, char *dimlist,  
                         char *maxdimlist, hid_t ntype, int merge)
```

| | |
|-------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Name of field to be defined |
| <i>dimlist</i> | IN: The list of geolocation dimensions defining the field |
| <i>maxdimlist</i> | IN: The maximum dimension list of geolocation dimensions defining the field |
| <i>ntype</i> | IN: The number type of the data stored in the field |
| <i>merge</i> | IN: Merge code (HE5_HDFE_NOMERGE (0) - no merge, HE5_HDFE_AUTOMERGE(1) - merge |

Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”

| | |
|--------------|--|
| Purpose | Defines a new geolocation field within the swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list. |
| Description | This routine defines geolocation fields to be stored in the swath. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first. The API will attempt to merge into a single object those fields that share dimensions and in case of multidimensional fields, numbertype. Two and three dimensional fields will be merged into a single three-dimensional object if the last two dimensions (in C order are equal). If the merge code for a field is set to 0, the API will not attempt to merge it with other fields. Fields using the unlimited dimension will not be merged. |
| Example | In this example, we define the geolocation fields, <i>Longitude</i> and <i>Latitude</i> with dimensions <i>GeoTrack</i> and <i>GeoXtrack</i> and containing 4 byte floating point numbers. We allow these fields to be merged into a single object: |

```
status = HE5_SWdefgeofield(swathID, "Longitude",  
                           "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT, 0);
```

```

status = HE5_SWdefgeofield(swathID, "Latitude",
                           "GeoTrack,GeoXtrack", NULL, H5T_NATIVE_FLOAT,
                           HE5_HDFE_NOMERGE);

```

Note: To assure that the fields defined by *HE5_SWdefgeofield* are properly established in the file, the swath should be detached (and then reattached) before writing to any fields.

FORTRAN

```

integer function he5_swdefgfld(swathid, fieldname, dimlist, maxdimlist,
                               ntype, merge)
  integer          swathid
  character*(*)   fieldname
  character*(*)   dimlist
  character*(*)   maxdimlist
  integer          ntype
  integer          merge

```

The equivalent *FORTRAN* code for the first example above is:

```

parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_NOMERGE=0)

status=he5_swdefgfld(swathid,"Longitude","GeoXtrack,GeoTrack
", " ", HE5_HDFE_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

```

The dimensions are entered in *FORTRAN* order with the first dimension incremented first.

Define Indexed Mapping between Geolocation and Data Dimension

HE5_SWdefidxmap

herr_t HE5_SWdefidxmap(hid_t *swathID*, char **geodim*, char **datadim*, long *index*[]),

| | |
|----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>geodim</i> | IN: Geolocation dimension name |
| <i>datadim</i> | IN: Data dimension name |
| <i>index</i> | IN: The array containing the indices of the data dimension to which each geolocation element corresponds. |
| Purpose | Defines a non-regular mapping between the geolocation and data dimension. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is incorrect geolocation or data dimension name. |
| Description | If there does not exist a regular (linear) mapping between a geolocation and data dimension, then the mapping must be made explicit. Each element of the index array, whose dimension is given by the geolocation size, contains the element number (0-based) of the corresponding data dimension. |
| Example | In this example, we consider the (simple) case of a geolocation dimension, <i>IdxGeo</i> of size 5 and a data dimension <i>IdxData</i> of size 8. <pre>long index[5] = {0,2,3,6,7}; status = HE5_SWdefidxmap(swathID, "IdxGeo", "IdxData", index);</pre> In this case the 0th element of <i>IdxGeo</i> will correspond to the 0th element of <i>IdxData</i> , the 1st element of <i>IdxGeo</i> to the 2nd element of <i>IdxData</i> , etc. |
| FORTRAN | integer function he5_swdefimap(<i>swathid</i> , <i>geodim</i> , <i>datadim</i> , <i>index</i>) integer <i>swathid</i> character*(*) <i>geodim</i> character*(*) <i>datadim</i> integer*4 <i>index</i> (*) |

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swdefimap(swathid, "IdxGeo", "IdxData", index)
```

Define a Time Period of Interest

HE5_SWdeftimeperiod

hid_t HE5_SWdeftimeperiod(hid_t *swathID*, double *starttime* , double *stopTime*, int *mode*)

| | | |
|------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>starttime</i> | IN: | Start time of period |
| <i>stopTime</i> | IN: | Stop time of period |
| <i>mode</i> | IN: | Cross Track inclusion mode |
| Purpose | | Defines a time period for a swath. |
| Return value | | Returns the swath period ID if successful or FAIL (-1) otherwise. |
| Description | | This routine defines a time period for a swath. It returns a swath period ID which is used by the <i>HE5_SWextractperiod</i> routine to read all the entries of a data field within the time period. A cross track is within a time period if 1) its midpoint is within the time period "box", or 2) either of its endpoints is within the time period "box", or 3) any point of the cross track is within the time period "box", depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the time period even though a particular element of the cross track might be outside the time period. The swath structure must have the <i>Time</i> field defined |
| Example | | In this example, we define a time period with a start time of 35232487.2 and a stop time of 36609898.1. We will consider a cross track to be within the time period if either one of the time values at the endpoints of a cross track are within the time period. starttime = 35232487.2; stoptime = 36609898.1; periodID = HE5_SWdeftimeperiod(swathID, starttime, stoptime, HE5_HDFE_ENDPOINT); |

```
FORTRAN    integer function he5_swdeftmeper(swathid, starttime, stoptime, mode)
            integer      swathid
            real*8       starttime
            real*8       stoptime
            integer      mode
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_ENDPOINT=1)
starttime = 35232487.2
stoptime = 36609898.1
periodID = he5_swdeftmeper(swathID, starttime, stoptime,
                           HE5_HDFE_ENDPOINT)
```

Define a Vertical Subset Region

HE5_SWdefvrtrregion

hid_t HE5_SWdefvrtrregion(hid_t *swathID*, hid_t *regionID*, char **vertObj*, double *range[]*)

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>regionID</i> | IN: Region (or period) id from previous subset call |
| <i>vertObj</i> | IN: Dimension or field to subset by |
| <i>range</i> | IN: Minimum and maximum range for subset |
| Purpose | Subsets on a monotonic field or contiguous elements of a dimension. |
| Return value | Returns the swath region ID if successful or FAIL (-1) otherwise. |
| Description | Whereas the <i>HE5_SWdefboxregion</i> and <i>HE5_SWdeftimeperiod</i> routines perform subsetting along the “Track” dimension, this routine allows the user to subset along any dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range(case 2). In the second case, the field must be one-dimensional and the values must be monotonic (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT, LONG, FLOAT, DOUBLE.) This routine may be called after <i>HE5_SWdefboxregion</i> or <i>HE5_SWdeftimeperiod</i> to provide both geographic or time and “vertical” subsetting . In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the region ID to HE5_HDFE_NOPREVSUB (-1). |

This routine may be called up to eight times with the same region ID. It this way a region can be subsetted along a number of dimensions.

The *HE5_SWregioninfo* and *HE5_SWextractregion* routines work as before, however because there is no mapping performed between geolocation dimensions and data dimensions the field to be subsetted, (the field specified in the call to *HE5_SWregioninfo* and *HE5_SWextractregion*) must contain the dimension used explicitly in the call to *HE5_SWdefvrtrregion* (case 1) or the dimension of the one-dimensional field (case 2).

Example Suppose we have a field called *Pressure* of dimension *Height* (= 10) whose values increase from 100 to 1000. If we desire all the elements with values between 500 and 800, we make the call:

```
range[0] = 500.;  
range[1] = 800.;  
regionID = HE5_SWdefvrtrregion(swathID, HE5_HDFE_NOPREVSUB,  
"Pressure", range);
```

The routine determines the elements in the *Height* dimension which correspond to the values of the *Pressure* field between 500 and 800.

If we wish to specify the subset as elements 2 through 5 (0 - based) of the *Height* dimension, the call would be:

```
range[0] = 2;  
range[1] = 5;  
regionID = HE5_SWdefvrtrregion(swathID, HE5_HDFE_NOPREVSUB,  
"DIM:Height", range);
```

The “DIM:” prefix tells the routine that the range corresponds to elements of a dimension rather than values of a field.

In this example, any field to be subsetted must contain the *Height* dimension.

If a previous subset region or period was defined with id, *subsetID*, that we wish to refine further with the vertical subsetting defined above we make the call:

```
regionID = HE5_SWdefvrtrregion(swathID, subsetID, "Pressure",  
range);
```

The return value, *regionID* is set equal to *subsetID*. That is, the subset region is modified rather than a new one created.

We can further refine the subset region with another call to the routine:

```
freq[0] = 1540.3;  
freq[1] = 1652.8;  
  
regionID = HE5_SWdefvrtrregion(swathID, regionID,  
"FreqRange", freq);
```

```
FORTRAN    integer function he5_swdefvrtrg(swathid, regionid, vertobj, range)
            integer      swathid
            integer      regionid
            character(*) vertobj
            real*8       range(*)
```

The equivalent *FORTRAN* code for the examples above is:

```
parameter (HE5_HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionid = he5_swdefvrtrg(swathid, HE5_HDFE_NOPREVSUB,
"Pressure", range)
```

Detach from a Swath Structure

HE5_SWdetach

herr_t HE5_SWdetach(hid_t *swathID*)

| | |
|----------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| Purpose | Detaches from swath interface. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine should be run before exiting from the swath file for every swath opened by <i>HE5_SWcreate</i> or <i>HE5_SWattach</i> . |

Example In this example, we detach the swath structure, *ExampleSwath*:

```
status = HE5_SWdetach(swathID);
```

FORTRAN integer function he5_swdetach(swathid)

```
integer         swathid
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swdetach(swathid)
```

Retrieve Size of Specified Dimension

HE5_SWdiminfo

hsize_t HE5_SWdiminfo(hid_t *swathID*, char **dimname*)

| | |
|----------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>dimname</i> | IN: Dimension name |
| Purpose | Retrieve size of specified dimension. |
| Return value | Size of dimension if successful or FAIL (-1) otherwise. If -1, could signify an improper swath ID or dimension name. |

Description This routine retrieves the size of specified dimension.

Example In this example, we retrieve information about the dimension, "GeoTrack":

```
dimsize = HE5_SWdiminfo(swathID, "GeoTrack");
```

The return value, *dimsize*, will be equal to 2000.

FORTRAN

```
integer*4 function he5_swdiminfo(swathid,dimname)
integer           swathid
character*(*)   dimname
integer*4         dimsize
```

The equivalent *FORTRAN* code for the example above is:

```
dimsize = he5_swdiminfo(swathid, "GeoTrack")
```

Remove an Alias for Swath Data Field

HE5_SWdropalias

herr_t HE5_SWdropalias(hid_t *swathID*, int *fldgroup*, const char **aliasname*)

| | | |
|------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fldgroup</i> | IN: | Field group flag |
| <i>aliasname</i> | IN: | Name of alias to remove |
| Purpose | | Remove an alias for Swath data field |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | Removes alias associated with a Swath data field. |
| Example | | In this example, we create and alias for the data field <i>Temperature</i> . strcpy(aliasname, "temps 0 to 30"); status = HE5_SWdropalias(swathID, HE5_HDFE_DATAGROUP, aliasname); |

FORTRAN integer function he5_swdropalias (*swathid*, *fldgroup*, *aliasname*)
 integer *swathid*
 character(*) *fldgroup*
 character(*) *aliasname*

The equivalent *FORTRAN* code for the first example above is:

```
aliasname = "temps 0 to 30"  
  
status = he5_swdropalias(swathid, HE5_HDFE_DATAGROUP,  
aliasname)
```

Duplicate a Region or Period

HE5_SWdupregion

hid_t HE5_SWdupregion(hid_t *regionID*)

| | |
|-----------------|---|
| <i>regionID</i> | IN: Region or period ID returned by HE5_SWdefboxregion, HE5_SWdeftimeperiod, or HE5_SWdefvrtregion. |
| Purpose | Duplicates a region. |
| Return value | Returns new region or period ID if successful or FAIL (-1) otherwise. |
| Description | This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways. |
| Example | In this example, we first subset a swath with <i>HE5_SWdefboxregion</i> , duplicate the region creating a new region ID, <i>regionID2</i> , and then perform two different vertical subsets of these (identical) geographic subset regions: |

```
regionID = HE5_SWdefboxregion(swathID, cornerlon, cornerlat,  
                               HE5_HDFE_MIDPOINT);  
  
regionID2 = HE5_SWdupregion(regionID);  
  
regionID = HE5_SWdefvrtregion(swathID, regionID, "Pressure",  
                               rangePres);  
  
regionID2 = HE5_SWdefvrtregion(swathID, regionID2,  
                               "Temperature", rangeTemp);
```

FORTRAN integer he5_swdupreg(*regionid*)
 integer *regionid*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_MIDPOINT=0)  
  
regionid = he5_swdefboxreg(swathid, cornerlon, cornerlat,  
                           HE5_HDFE_MIDPOINT)  
  
regionid2 = he5_swdupreg(regionid)  
  
regionid = he5_swdefvrtreg(swathid, regionid, 'Pressure',  
                           rangePres)
```

```
regionid2 = he5_swdefvrtrg(swathid, regionid2,  
'Temperature', rangeTemp)
```

Read Data from a Defined Time Period

HE5_SWextractperiod

```
herr_t HE5_SWextractperiod(hid_t swathID, hid_t periodID, char *fieldname, int  
                           externalflag, void *buffer)
```

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach

periodID IN: Period id returned by HE5_SWdeftimeperiod

fieldname IN: Field to subset

externalflag IN: External geolocation mode

buffer OUT: Data buffer

Purpose Extracts (reads) from subsetted time period.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

NOTE: **External file functionality not available in this release**

Description This routine reads data into the data buffer from the subsetted time period. Only complete crosstracks are extracted. If the external_mode flag is set to *HE5_HDFE_EXTERNAL* (1) then the geolocation fields and the data field can be in different swaths. If set to *HE5_HDFE_INTERNAL* (0), then these fields must be in the same swath structure.

Example In this example, we read data within the subsetted time period defined in *HE5_SWdeftimeperiod* from the *Spectra* field. Both the geolocation fields and the *Spectra* data field are in the same swath.

```
status = HE5_SWextractperiod(SwathID, periodID, "Spectra",  
                           HE5_HDFE_INTERNAL, datbuf);
```

FORTRAN integer function he5_swextper(*swathID*, *periodid*, *fieldname*, *externalflag*,
buffer)

integer *swathID*

integer *periodid*

character*(*) *fieldname*

integer *externalflag*

<valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_INTERNAL=0)

status = he5_swextper(swathid, periodid, "Spectra",
HE5_HDFE_INTERNAL, datbuf)
```

Read Data from a Geographic Region

HE5_SWextractregion

```
herr_t HE5_SWextractregion(hid_t swathID, hid_t regionID, char *fieldname, int  
                           externalflag, void *buffer)
```

| | | |
|---------------------|------|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>regionID</i> | IN: | Region ID returned by HE5_SWdefboxregion |
| <i>fieldname</i> | IN: | Field to subset |
| <i>externalflag</i> | IN: | External geolocation mode |
| <i>buffer</i> | OUT: | Data buffer |

Purpose Extracts (reads) from subsetted region.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

NOTE: **External file functionality not available in this release**

Description This routine reads data into the data buffer from the subsetted region. Only complete crosstracks are extracted. If the external_mode flag is set to *HE5_HDFE_EXTERNAL* (1) then the geolocation fields and the data field can be in different swaths. If set to *HE5_HDFE_INTERNAL* (0), then these fields must be in the same swath structure.

Example In this example, we read data within the subsetted region defined in *HE5_SWdefboxregion* from the *Spectra* field. Both the geolocation fields and the *Spectra* data field are in the same swath.

```
status = HE5_SWextractregion(SWid, regionID, "Spectra",  
                           HE5_HDFE_INTERNAL, datbuf);
```

FORTRAN integer function he5_swextreg(*swathid*, *regionid*, *fieldname*, *externalflag*,
 buffer)

 integer *swathid*
 integer *regionid*
 character(*) *fieldname*
 integer *externalflag*
 <valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_INTERNAL=0)

status = he5_swextreg(swathid, regionid, "Spectra",
                      HE5_HDFE_INTERNAL, datbuf)
```

Retrieve Information about a Swath Field

HE5_SWfieldinfo

```
herr_t HE5_SWfieldinfo(hid_t swathID, char *fieldname, int *rank, hsize_t  
                      dims[], H5T_class_t ntype[], char *dimlist, char *maxdimlist)
```

| | |
|-------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Fieldname |
| <i>rank</i> | OUT: Rank of field |
| <i>dims</i> | OUT: Array containing the dimension sizes of the field |
| <i>ntype</i> | OUT: Array containing the numbertype of the field |
| <i>dimlist</i> | OUT: List of dimensions in field |
| <i>maxdimlist</i> | OUT: List of maximum dimensions in field |
| Purpose | Retrieve information about a specific geolocation or data field in the swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified field does not exist. |
| Description | This routine retrieves information on a specific data field. |
| Example | In this example, we retrieve information about the <i>Spectra</i> data fields: |

```
status = HE5_SWfieldinfo(swathID, "Spectra", &rank, dims,  
                         numbertype, dimlist, maxdimlist);
```

The return parameters will have the following values:

rank=3, *numbertype*=5, *dims*[3]={5,4000,2000} and *dimlist*="Bands, DataTrack, DataXtrack"

If one of the dimensions in the field is appendable, then the current value for that dimension will be returned in the *dims* array.

FORTRAN

```

integer function he5_swfldinfo(swathid, fieldname, rank, dims, ntype,
dimlist, maxdimlist)

integer          swathid
character(*)    fieldname
integer          rank
integer*4        dims(*)
integer          ntype(*)
character(*)    dimlist
character(*)    maxdimlist

```

The equivalent *FORTRAN* code for the example above is:

```

status = he5_swfldinfo(swathid, "Spectra", rank, dims,
numbertype, dimlist, maxdimlist)

```

The return parameters will have the following values:

*rank=3, numbertype=5, dims[3]={2000,4000,5} and
dimlist="DataXtrack, DataTrack, Bands"*

Note that the dimensions array and dimension list are in *FORTRAN* order.

Rename Swath Data Field

HE5_SWfldrename

```
herr_t HE5_SWfldrename(hid_t swathID, char *oldfieldname, const char  
*newfieldname)
```

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach

oldfieldname IN: Current name of field

newfieldname IN: New name of field

Purpose Rename swath data field

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This function allows the user to change the name of a field. This is useful in case the user would want to update the data field to reflect a version change in the calibration of a data field and show that in the name of the field.

Example In this example, we create and alias for the data field *Temperature*.

```
strcpy(newfieldname, "temps 0 to 30");  
  
status = HE5_SWfldrename(swathID, "Temperature",  
newfieldname);
```

FORTRAN integer function he5_swfldrename (*swathid*, *oldfieldname*, *newfieldname*)
integer *swathid*
character(*) *oldfieldname*
character(*) *newfieldname*

The equivalent *FORTRAN* code for the first example above is:

```
newfieldname = "temps 0 to 30"  
  
status = he5_swfldrename(swathid, "Temperature",  
newfieldname)
```

Get External Data File Information

HE5_SWgetextdata

```
int HE5_SWgetextdata(hid_t swathID, char *fieldname, size_t namelength, char  
*filelist, off_t offset[], hsize_t size[])
```

| | |
|-------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: External field name |
| <i>namelength</i> | OUT: Length of each name entry |
| <i>filelist</i> | OUT: List of file names |
| <i>offset</i> [] | OUT: Array of offsets (in byte) from the beginning of file to the location in file where the data starts |
| <i>size</i> [] | OUT: Array of sizes (in bytes) reserved in the file for the data |
| Purpose | Retrieves information about external data file(s) associated with the data set. |
| Return value | Returns number of external data files if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID or field name. |
| Example | In this example, we get information about the <i>ExtData</i> field: |
| | <pre>nfiles = HE5_SWgetextdata(swathID, "ExtData", namlen, filenames, offset, size);</pre> |
| FORTRAN | <pre>integer function he5_swgetxdat(swathid,fieldname,nlen,flist,offset, size) integer swathid integer nfiles integer*4 nlen integer*4 offset(*) integer*4 size(*) character*(*) fieldname character*(*) flist</pre> |

The equivalent *FORTRAN* code for the example above is:

```
nfiles = he5_swgetxdat(swathid, "ExtData",nlen,  
flist,offset,size)
```

Get Fill Value for a Specified Field

HE5_SWgetfillvalue

herr_t HE5_SWgetfillvalue(hid_t *swathID*, char **fieldname*, void **fillval*)

| | | |
|------------------|------|--|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: | Fieldname |
| <i>fillval</i> | OUT: | Space allocated to store the fill value |
| Purpose | | Retrieves fill value for the specified field. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type. |
| Description | | It is assumed the number type of the fill value is the same as the field. |
| Example | | In this example, we get the fill value for the <i>Temperature</i> field: |

```
status = HE5_SWgetfillvalue(swathID, "Temperature",  
&tempfill);
```

FORTRAN integer function he5_swgetfill(*swathid,fieldname,fillval*)
 integer *swathid*
 character*(*) *fieldname*
 <valid type> *fillval(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swgetfill(swathid, "Temperature", tempfill)
```

Retrieve Type of Dimension Mapping when first Dimension is Geodim

HE5_SWgeomapinfo

herr_t HE5_SWgeomapinfo(hid_t *swathID*, char **geodim*)

| | |
|----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>geodim</i> | IN: Dimension name |
| Purpose | Retrieve type of dimension mapping for a dimension. |
| Return value | Returns (2) for indexed mapping, (1) for regular mapping, (0) if dimension is not mapped, or FAIL (-1) otherwise. |
| Description | This routine checks the type of mapping (regular or indexed). |
| Example | In this example, we retrieve information about the type of mapping between the “IdxGeo” and “IdxData” dimensions, defined by <i>HE5_SWdefidxmap</i> . |

```
status = HE5_SWgeomapinfo(swathID, geodim);
```

We will have regmap = 2 for indexed mapping between the “IdxGeo” and “IdxData” dimensions.

NOTE: If the dimension has regular mapping and indexed, the function will return a value of 3.

| | |
|---------|---|
| FORTRAN | integer function he5_swgmapinfo(<i>swathid</i> , <i>geodim</i>) |
| | integer <i>swathid</i> |
| | character*(*) <i>geodim</i> |

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swgmapinfo(swathid, geodim)
```

Return Information about a Group Swath Attribute

HE5_SWgrpattrinfo

```
herr_t HE5_SWgrpattrinfo(hid_t swathID, const char *attrname, H5T_class_t  
*ntype, hsize_t *count)
```

| | |
|-------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: Attribute name |
| <i>numbertype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of attribute elements |
| Purpose | Returns information about a swath group attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a swath group attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_SWgrpattrinfo(swathID, "ScalarFloat", &nt,  
&count);
```

The *nt* variable will have the value 1 and *count* will have the value 1.

FORTRAN

```
integer function he5_swgattrinfo(swathid, attrname, ntype, count,  
integer      swathid  
character(*) attrname  
integer      ntype  
integer *4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_swgattrinfo(swathid, "ScalarFloat", nt, count)
```

Retrieve Indexed Geolocation Mapping

HE5_SWidxmapinfo

```
hsize_t HE5_SWidxmapinfo(hid_t swathID, char *geodim, char *datadim, long  
                           index[])
```

| | |
|----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>geodim</i> | IN: Indexed Geolocation dimension name |
| <i>datadim</i> | IN: Indexed Data dimension name |
| <i>index</i> | OUT: Index mapping array |
| Purpose | Retrieve indexed array of specified geolocation mapping. |
| Return value | Returns size of indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist. |
| Description | This routine retrieves the size of the indexed array and the array of indexed elements of the specified geolocation mapping. |
| Example | In this example, we retrieve information about the indexed mapping between the "IdxGeo" and "IdxData" dimensions: |

```
idxsz = HE5_SWidxmapinfo(swathID, "IdxGeo", "IdxData",  
                           index);
```

The variable, *idxsz*, will be equal to 5 and *index[5]* = {0,2,3,6,7}.

FORTRAN

```
integer*4 function he5_swimapinfo(swathid, geodim, datadim, index)  
  integer         swathid  
  character*(*)   geodim  
  character*(*)   datadim  
  integer*4        index(*)
```

The equivalent *FORTRAN* code for the example above is:

```
idxsz = he5_swimapinfo(swathid, "IdxGeo", "IdxData", index)
```

Retrieve Information Swath Attributes

HE5_SWinqattrs

long HE5_SWinqattrs(hid_t *swathID*, char **attrnames*, long **strbufsize*)

| | |
|-------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about attributes defined in swath. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the attributes defined in a swath structure. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = HE5_SWinqattrs(swathID, NULL, &strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = HE5_SWinqattrs(swathID, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to:

"*attrOne,attr_2*".

FORTRAN integer*4 function he5_swinqattrs(*swathid,attrnames,strbufsize*)
integer *swathid*
character(*) *attrnames*
integer*4 *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_swingattrs(swathid, attrnames, strbufsize)
```

Retrieve Information about Data Fields Defined in Swath

HE5_SWinqdatafields

```
long HE5_SWinqdatafields(hid_t swathID, char *fieldlist, int rank[], H5T_class_t ntype[])
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldlist</i> | OUT: Listing of data fields (entries separated by commas) |
| <i>rank</i> | OUT: Array containing the rank of each data field |
| <i>ntype</i> | OUT: Array containing the numbertype of each data field |
| Purpose | Retrieve information about all of the data fields defined in swath. |
| Return value | Number of data fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id. |
| Description | The field list is returned as a string with each data field separated by commas. The <i>rank</i> and <i>ntype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned. |
| Example | In this example we retrieve information about the data fields: |

```
nflds = HE5_SWinqdatafields(swathID, fieldlist, rank,  
                             ntype);
```

The parameter, *fieldlist*, will have the value:

"Spectra" with *rank*[1]={3}, *ntype*[1]={1}

FORTRAN integer*4 function he5_swinqdflds(*swathid*, *fieldlist*, *rank*, *ntype*)
 integer *swathid*
 character(*) *fieldlist*
 integer *rank*(*)
 integer *ntype*(*)

The equivalent *FORTRAN* code for the example above is:

```
nflds = he5_swinqdflds(swathid, fieldlist, rank, ntype)
```

Retrieve Information about Dimensions Defined in Swath

HE5_SWinqdims

long HE5_SWinqdims(hid_t *swathID*, char **dimnames*, hsize_t *dims*[])

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>dimnames</i> | OUT: Dimension list (entries separated by commas) |
| <i>dims</i> | OUT: Array containing size of each dimension |
| Purpose | Retrieve information about all of the dimensions defined in swath. |
| Return value | Number of dimension entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id. |
| Description | The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to <i>NULL</i> will not be returned. |
| Example | In this example, we retrieve information about the dimensions defined in the <i>ExampleSwath</i> structure: |

```
ndims = HE5_SWinqdims(swathID, dimnames, dims);
```

The parameter, *dimname*, will have the value:

"GeoTrack,GeoXtrack,DataTrack,DataXtrack,Bands,Unlim"

with *ndims* = 6, *dims*[6]={2000,1000,4000,2000,5,0}

FORTRAN integer*4 function he5_swinqdims(*swathid*,*dimnames*,*dims*)
 integer *swathid*
 character*(*) *dimnames*
 integer*4 *dims*(*)

The equivalent *FORTRAN* code for the example above is:

```
ndims = he5_swingdims(swathid, dimnames, dims)
```

Retrieve Information about Geolocation Fields Defined in Swath

HE5_SWinqgeofields

```
long HE5_SWinqgeofields(hid_t swathID, char *fieldlist, int rank[], H5T_class_t ntype[])
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldlist</i> | OUT: Listing of geolocation fields (entries separated by commas) |
| <i>rank</i> | OUT: Array containing the rank of each geolocation field |
| <i>ntype</i> | OUT: Array containing the numbertype of each geolocation field |
| Purpose | Retrieve information about all of the geolocation fields defined in swath. |
| Return value | Number of geolocation fields found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id. |
| Description | The field list is returned as a string with each geolocation field separated by commas. The <i>rank</i> and <i>ntype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned. |
| Example | In this example, we retrieve information about the geolocation fields: |

```
nflds = HE5_SWinqgeofields(swathID, fieldlist, rank,  
                           ntype);
```

The parameter, *fieldlist*, will have the value: "Longitude,Latitude" with
nflds = 2, *rank*[2]={2,2}, *ntype*[2]={1,1}

FORTRAN

```
integer*4 function he5_swinqgfls(swathid, fieldlist, rank, ntype)  
integer          swathid  
character(*)    fieldlist  
integer         rank(*)  
integer         ntype(*)
```

The equivalent *FORTRAN* code for the example above is:

```
nflds = he5_swinqgfls(swathid, fieldlist, rank, ntype)
```

Retrieve Information about Swath Group Attributes

HE5_SWinqgrpattrs

long HE5_SWinqgrpattrs(hid_t *swathID*, char **attrnames*, long **strbufsize*)

| | |
|-------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about group attributes defined in swath. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each group attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the group attributes defined for the “Data Fields” group. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = HE5_SWinqgrpattrs(swathID, NULL, &strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = HE5_SWinqgrpattrs(swathID, attrnames, &strbufsize);
```

The variable, *attrnames*, will be set to:

“*attrOne,attr_2*”.

FORTRAN integer*4 function he5_swinqgatrs(*swathid*,*attrnames*,*strbufsize*)

integer *swathid*

character*(*) *attrnames*

integer*4 *strbufsize*

integer*4 *nattr*

The equivalent **FORTRAN** code for the example above is:

```
nattr = he5_swinqgatrs(swathid, attrnames, strbufsize)
```

Retrieve Information about Indexed Mappings Defined in Swath

HE5_SWinqidxmaps

long HE5_SWinqidxmaps(hid_t *swathID*, char **idxmap*, hsize_t *idxsizes*[])

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>idxmap</i> | OUT: Indexed Dimension mapping list (entries separated by commas) |
| <i>idxsizes</i> | OUT: Array containing the sizes of the corresponding index arrays. |
| Purpose | Retrieve information about all of the indexed geolocation/data mappings defined in swath. |
| Return value | Number of indexed mapping relations found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper Swath ID. |
| Description | The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to <i>NULL</i> , will not be returned. |
| Example | In this example, we retrieve information about the indexed dimension mappings: |

```
nidxmaps = HE5_SWinqidxmaps(swathID, idxmap, idxsizes);
```

The variable, *idxmap*, will contain the string:

"IdxGeo/IdxData" with *nidxmaps* = 1 and *idxsizes*[1]={5}.

FORTRAN integer*4 function he5_swinqimaps(*swathid*,*dimmap*,*idxsizes*)
integer *swathid*
character(*) *dimmap*
integer*4 *idxsizes*(*)

The equivalent *FORTRAN* code for the example above is:

```
nidxmaps = he5_swinqimaps(swathid, dimmap, idxsizes)
```

Retrieve Information Swath Local Attributes

HE5_SWinqlocattrs

```
long HE5_SWinqlocattrs(hid_t swathID, const char *fieldname, char *attrnames,  
                        long *strbufsize)
```

| | |
|-------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Fieldname to retrieve local attribute information |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about local attributes defined for a field. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each local attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |
| Example | In this example, we retrieve information about the local attributes defined for a field “DataField”.. We assume that there are two attributes stored, <i>attrOne</i> and <i>attr_2</i> : |
| | <pre>nattr = HE5_SWinqlocattrs(swathID, "DataField", NULL, &strbufsize);</pre> |
| | The parameter, <i>nattr</i> , will have the value 2 and <i>strbufsize</i> will have value 14. |
| | <pre>nattr = HE5_SWinqlocattrs(swathID, "DataField", attrnames, &strbufsize);</pre> |
| | The variable, <i>attrlist</i> , will be set to: "attrOne,attr_2". |
| FORTRAN | <pre>integer*4 function he5_swinqlatrs(swathid ,fieldname, attrnames, strbufsize) integer swathid character(*) fieldname character(*) attrnames</pre> |

```
integer*4      strbufsize
```

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_swinglattr(swatchid, "DataField", attrnames,  
strbufsize)
```

Retrieve Information about Dimension Mappings Defined in Swath

HE5_SWinqmaps

```
long HE5_SWinqmaps(hid_t swathID, char *dimmap, long offset[], long increment[])
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>dimmap</i> | OUT: Dimension mapping list (entries separated by commas) |
| <i>offset</i> | OUT: Array containing the offset of each geolocation relation |
| <i>increment</i> | OUT: Array containing the increment of each geolocation relation |
| Purpose | Retrieve information about all of the (non-indexed) geolocation relations defined in swath. |
| Return value | Number of geolocation relation entries found if successful or FAIL (-1) otherwise. A typical reason for failure is an improper Swath ID. |
| Description | The dimension mapping list is returned as a string with each mapping separated by commas. The two dimensions in each mapping are separated by a slash (/). Output parameters set to <i>NULL</i> will not be returned. |
| Example | In this example, we retrieve information about the dimension mappings in the <i>ExampleSwath</i> structure: |

```
nmaps = HE5_SWinqmaps(swathID, dimmap, offset, increment);
```

The variable, *dimmap*, will contain the string:

"GeoTrack/DataTrack,GeoXtrack/DataXtrack" with *nmaps* = 2, *offset*[2]={0,1} and *increment*[2]={2,2}.

FORTRAN

integer*4 function

```
he5_swinqmaps(swathid, dimmap, offset, increment)
```

integer *swathid*

character*(*) *dimmap*

integer*4 *offset*(*)

integer*4 *increment*(*)

The equivalent *FORTRAN* code for the example above is:

```
nmaps = he5_swinqmaps(swathid, dimmap, offset, increment)
```

Retrieve Swath Structures Defined in HDF-EOS File

HE5_SWinqswath

long HE5_SWinqswath(const char * *filename*, char **swathlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>filename</i> | IN: The HDF-EOS file name |
| <i>swathlist</i> | OUT: Swath list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of swath list |
| Purpose | Retrieves number and names of swaths defined in HDF-EOS file. |
| Return value | Number of swaths found if successful or FAIL (-1) otherwise. |
| Description | The swath list is returned as a string with each swath name separated by commas. If <i>swathlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . If <i>strbufsize</i> is also set to NULL, the routine returns just the number of swaths. Note that <i>strbufsize</i> does not count the null string terminator. |
| Example | In this example, we retrieve information about the swaths defined in an HDF-EOS file, <i>Swath.he5</i> . We assume that there are two swaths stored, <i>SwathOne</i> and <i>Swath_2</i> : <code>nswath = HE5_SWinqswath("Swath.he5", NULL, &strbufsize);</code> The parameter, <i>nswath</i> , will have the value 2 and <i>strbufsize</i> will have value 16. <code>nswath = HE5_SWinqswath("Swath.he5", swathlist, &strbufsize);</code> The variable, <i>swathlist</i> , will be set to: “ <i>SwathOne,Swath_2</i> ”. |
| FORTRAN | integer*4 function he5_swingswath(<i>filename</i> , <i>swathlist</i> , <i>strbufsize</i>) character*(*) <i>filename</i> character*(*) <i>swathlist</i> integer*4 <i>strbufsize</i> The equivalent FORTRAN code for the example above is: <code>nswath = he5_swingswath('Swath.he5', swathlist, strbufsize)</code> |

Return Information about a Local Swath Attribute

HE5_SWlocattrinfo

```
herr_t HE5_SWlocattrinfo(hid_t swathID, const char *fieldname, const char  
*attrname, H5T_class_t *ntype, hsize_t *count)
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Field name |
| <i>attrname</i> | OUT: Attribute name |
| <i>ntype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of attribute elements |
| Purpose | Returns information about a Data Field's local attribute(s) |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a data field's local attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_SWlocattrinfo(swathID, "DataField", attrname,  
&ntype, &count);
```

The *nt* variable will have the value 1 and *count* will have the value 1.

FORTRAN

```
integer function he5_swlattrinfo(swathid, fieldname, attrname, ntype,  
count)  
integer      swathid  
character(*) attrname  
integer      ntype  
integer *4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_swlattrinfo(swathid, "DataField", attrname,  
ntype, count)
```

Retrieve Offset and Increment of Specific Dimension Mapping

HE5_SWmapinfo

```
herr_t HE5_SWmapinfo(hid_t swathID, char *geodim, char *datadim, long  
                      *offset, long *increment)
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>geodim</i> | IN: Geolocation dimension name |
| <i>datadim</i> | IN: Data dimension name |
| <i>offset</i> | OUT: Mapping offset |
| <i>increment</i> | OUT: Mapping increment |
| Purpose | Retrieve offset and increment of specific monotonic geolocation mapping. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist. |
| Description | This routine retrieves offset and increment of the specified geolocation mapping. |
| Example | In this example, we retrieve information about the mapping between the <i>GeoTrack</i> and <i>DataTrack</i> dimensions: <pre>status = HE5_SWmapinfo(swathID, "GeoTrack", "DataTrack", &offset, &increment);</pre> The variable <i>offset</i> will be 0 and <i>increment</i> 2. |
| FORTRAN | <pre>integer function he5_swmapinfo(<i>swathid</i>, <i>geodim</i>, <i>datadim</i>, <i>offset</i>, <i>increment</i>)</pre> <pre>integer <i>swathid</i> character(*) <i>geodim</i> character(*) <i>datadim</i> integer*4 <i>offset</i> integer*4 <i>increment</i></pre> |

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swmapinfo(swathid, "GeoTrack", "DataTrack",
                       offset, increment)
```

Mount External Data File

HE5_SWmountexternal

```
hid_t HE5_SWmountexternal(hid_t swathID, int fldgroup, const char  
*extfilename)
```

swathID IN: Swath ID returned by HE5_SWcreate or HE5_SWattach

fldgroup IN: Field group flag

extfilename IN: External file name

Purpose Mount external data file

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This function allows the user to store required data needed by multiple data files into a separate file so it is not repeated thoughtout the data files.

Example In this example, we mount a file that contains calibration information needed by the data fields in another file

```
strcpy(extfilename, "/home/user/data/calibration.hdf5");  
  
fileID = HE5_SWmountexternal(swathID, HE5_HDFE_DATAGROUP,  
extfilename);
```

FORTRAN Not available with this release.

Return Number of Specified Objects in a Swath

HE5_SWnentries

long HE5_SWnentries(hid_t *swathID*, int *entrycode*, long **strbufsize*)

| | |
|-------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>entrycode</i> | IN: Entrycode |
| <i>strbufsize</i> | OUT: String buffer size |
| Purpose | Returns number of entries and descriptive string buffer size for a specified entity. |
| Return value | Number of entries if successful or FAIL (-1) otherwise. A typical reason for failure is an improper swath id or entry code. |
| Description | This routine can be called before an inquiry routines in order to determine the sizes of the output arrays and descriptive strings. The string length does not include the NULL terminator. |

The entry codes are:

- HE5_HDFE_NENTDIM (0) - Dimensions
- HE5_HDFE_NENTMAP (1) - Dimension Mappings
- HE5_HDFE_NENTIMAP (2) - Indexed Dimension Mappings
- HE5_HDFE_NENTGFLD (3) - Geolocation Fields
- HE5_HDFE_NENTDFLD (4) - Data Fields

Example In this example, we determine the number of dimension mapping entries and the size of the map list string.

```
nmaps = HE5_SWnentries(swathID, HE5_HDFE_NENTMAP, &bufsize);
```

The return value, *nmaps*, will be equal to 2 and *bufsz* = 39

FORTRAN integer*4 function he5_swnentries(*swathid*, *entrycode*, *bufsize*)
integer *swathid*
integer *entrycode*
integer*4 *bufsize*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NENTMAP=1)  
nmaps = he5_swnentries(swathid, HE5_HDFE_NENTMAP, bufsize)
```

Open HDF-EOS File

HE5_SWopen

hid_t HE5_SWopen(const char *filename, uintn access)

| | |
|-----------------|--|
| <i>filename</i> | IN: Complete path and filename for the file to be opened |
| <i>access</i> | IN: H5F_ACC_RDONLY, H5F_ACC_RDWR or H5F_ACC_TRUNC |
| Purpose | Opens or creates HDF-EOS file in order to create, read, or write a Swath. |
| Return value | Returns the swath file id handle (<i>fid</i>) if successful or FAIL (-1) otherwise. |
| Description | This routine creates a new file or opens an existing one, depending on the access parameter. |

Access codes:

- H5F_ACC_RDONLY Open for read only. If file does not exist, error
- H5F_ACC_RDWR Open for read/write. If file does not exist, create it
- H5F_ACC_TRUNC If file exist, delete it, then open a new file for read/write

Example In this example, we create a new swath file named, *Swath.he5*. It returns the file handle, *fid*.

```
fid = HE5_SWopen( "Swath.he5" , H5F_ACC_TRUNC );  
FORTRAN integer function he5_swopen(filename, access)  
character*(*) filename  
integer access
```

The access codes should be defined as parameters:

```
parameter (HE5_HDFE_RDWR = 0)  
parameter (HE5_HDFE_RDONLY = 1)  
parameter (HE5_HDFE_TRUNC = 2)
```

The equivalent *FORTRAN* code for the example above is:

```
fid = he5_swopen( "Swath.he5" , HE5_HDFE_TRUNC )
```

Note to users of the SDP Toolkit: Please refer to the *Release 5B SDP Toolkit User Guide for the ECS Project (333-CD-510-001)*, Section 6.2.1.2, for information on how to obtain a file name (referred to as a “physical file handle”) from within a PGE. See also Section 9 of this document for code examples.

Return Information about a Defined Time Period

HE5_SWperiodinfo

```
herr_t HE5_SWperiodinfo(hid_t swathID, hid_t periodID, char * fieldname,  
                        H5T_class_t *ntype, int *rank, hsize_t dims[], size_t *size)
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>periodID</i> | IN: Period ID returned by HE5_SWdeftimeperiod |
| <i>fieldname</i> | IN: Field to subset |
| <i>ntype</i> | OUT: Number type of field |
| <i>rank</i> | OUT: Rank of field |
| <i>dims</i> | OUT: Dimensions of subset period |
| <i>size</i> | OUT: Size in bytes of subset period |
| Purpose | Retrieves information about the subsetted period. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns information about a subsetted time period for a particular field. It is useful when allocating space for a data buffer for the subset. Because of differences in number type and geolocation mapping, a given time period will give different values for the dimensions and size for various fields. |
| Example | In this example, we retrieve information about the time period defined in <i>HE5_SWdeftimeperiod</i> for the <i>Spectra</i> field. We use this to allocate space for data in the subsetted time period. <pre>/* Get size in bytes of time period for "Spectra" field*/ status = HE5_SWperiodinfo(SWid, periodID, "Spectra", &ntype, &rank, dims, &size); /* Allocate space */ datbuf = (double *)calloc(size, sizeof(double));</pre> |

FORTRAN integer function he5_swperinfo(*swathid, periodid, fieldname, ntype, rank, dims, size*)
 integer *swathid*
 integer *periodid*
 character*(*) *fieldname*
 integer *ntype*
 integer *rank*
 integer*4 *dims(*)*
 integer*4 *size*

The equivalent *FORTRAN* code for the example above is:

```
status=he5_swperinfo(swid,periodid,"Spectra",ntype,rank,dim,  
size)
```

Read Swath Attribute

HE5_SWreadattr

herr_t HE5_SWreadattr(hid_t *swathID*, const char **attrname*, void **datbuf*)

| | | |
|-----------------|------|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: | Attribute name |
| <i>datbuf</i> | OUT: | Buffer allocated to hold attribute values |
| Purpose | | Reads attribute from a swath. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type or incorrect attribute name. |
| Description | | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | | In this example, we read a floating point attribute with the name "ScalarFloat": |
| FORTRAN | | <pre>status = HE5_SWreadattr(swathID, "ScalarFloat", &data);</pre> <pre>integer function he5_swrdatr(swathid,attrname,datbuf)</pre> <pre>integer swathid</pre> <pre><valid type> datbuf(*)</pre> |

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swrdatr(swathid, "ScalarFloat", datbuf)
```

Read External Data Set

HE5_SWreadexternal

```
herr_t HE5_SWreadexternal(hid_t swathID, int fldgroup, const char *fieldname,  
                          void *buffer)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fldgroup</i> | IN: Field group flag |
| <i>fieldname</i> | IN: Name of field to read |
| <i>buffer</i> | OUT: Output data buffer |
| Purpose | Read external data set |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This function allows the user to get the data required from the external data file. |
| Example | In this example, the field “Cal data” is read from the external file: <pre>strcpy(<i>fieldname</i>, "Cal data"); status = HE5_SWreadexternal(<i>swathID</i>, HE5_HDFE_DATAGROUP, <i>fieldname</i>, <i>buffer</i>);</pre> |
| FORTRAN | Not available with this release. |

Read Data from a Swath Field

HE5_SWreadfield

```
herr_t HE5_SWreadfield(hid_t swathID, char *fieldname, const hsize_t start[],  
                      const hsize_t stride[], const hsize_t edge[], void *buffer)
```

| | | |
|------------------|---|--|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: | Name of field to read |
| <i>start</i> | IN: | Array specifying the starting location within each dimension |
| <i>stride</i> | IN: | Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: | Array specifying the number of values to read along each dimension |
| <i>buffer</i> | OUT: | Buffer to store the data read from the field |
| Purpose | Reads data from a swath field. | |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are improper swath id or unknown fieldname. | |
| Description | The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the swath field (input) dimensions. The output data in <i>buffer</i> is written to contiguously. The default values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are <i>(dim - start) / stride</i> where <i>dim</i> refers is the size of the dimension. | |
| Example | In this example, we read data from the 10th track (0-based) of the <i>Longitude</i> field. | |
| | <pre>float track[1000]; hssize_t start[2] = {9,1}; hsize_t edge[2] = {1,1000}; status = HE5_SWreadfield(swathID, "Longitude", start, NULL, edge, track);</pre> | |

```

FORTRAN integer function he5_swrdfld(swathid, fieldname, start, stride, edge,buffer)
          integer         swathid
          character*(*)   fieldname
          integer*4        start(*)
          integer*4        stride(*)
          integer*4        edge(*)
          <valid type>    buffer(*)

```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```

real*4      track(1000)

integer*4   start(2), stride(2), edge(2)

start(1) = 0
start(2) = 10
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 1

status=he5_swrdfld(swathid,"Longitude",start,stride,
edge,track)

```

Read Group Swath Attribute

HE5_SWreadgrpattr

herr_t HE5_SWreadgrpattr(hid_t *swathID*, const char **attrname*, void **dbuf*)

| | |
|-----------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: Attribute name |
| <i>dbuf</i> | OUT: Buffer allocated to hold attribute values |
| Purpose | Reads attribute from a swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read a floating point attribute with the name "ScalarFloat": |
| FORTRAN | <pre>status = HE5_SWreadgrpattr(swathID, "ScalarFloat", &data);</pre> <pre>integer function he5_swrdgattr(swathid,attrname,dbuf)</pre> <pre>integer swathid</pre> <pre>character*(*) attrname</pre> <pre><valid type>dbuf(*)</pre> |

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swrdgattr(swathid, "ScalarFloat", datbuf)
```

Read Local Swath Attribute

HE5_SWreadlocattr

```
herr_t HE5_SWreadlocattr(hid_t swathID, const char *fieldname, const char  
*attrname, void *datbuf)
```

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Field name |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | OUT: Buffer allocated to hold attribute values |
| Purpose | Reads attribute from a swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read a single precision (32 bit) floating point attribute with the name "ScalarFloat": |

```
status = HE5_SWreadlocattr(swathID, "DataField",  
"ScalarFloat", &data);
```

FORTRAN

```
integer function he5_swrdlattr(swathid, fieldname, attrname, datbuf)  
  
integer           swathid  
character(*)     fieldname  
character(*)     attrname  
<valid type>    datbuf(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swrdlattr(swathid, "DataField", "ScalarFloat",  
datbuf)
```

Define a Longitude-Latitude Box Region for a Swath

HE5_SWregionindex

hid_t HE5_SWregionindex(hid_t *swathID*, double *cornerlon*[], double *cornerlat*[], int *mode*, char **geodim*, hsize_t *idxrange*[])

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>cornerlon</i> | IN: Longitude in decimal degrees of box corners |
| <i>cornerlat</i> | IN: Latitude in decimal degrees of box corners |
| <i>mode</i> | IN: Cross Track inclusion mode |
| <i>geodim</i> | OUT: Geolocation track dimension |
| <i>idxrange</i> | OUT: The indices of the region in the geolocation track dimension. |
| Purpose | Defines a longitude-latitude box region for a swath. |
| Return value | Returns the swath region ID if successful or FAIL (-1) otherwise. |
| Description | The difference between this routine and <i>HE5_SWdefboxregion</i> is the geolocation track dimension name and the range of that dimension are returned in addition to a regionID. Other than that difference they are the same function and this function is used just like <i>HE5_SWdefboxregion</i> . This routine defines a longitude-latitude box region for a swath. It returns a swath region ID which is used by the <i>HE5_SWextractregion</i> routine to read all the entries of a data field within the region. A cross track is within a region if 1) its midpoint is within the longitude-latitude "box" (HE5_HDFE_MIDPOINT), or 2) either of its endpoints is within the longitude-latitude "box" (HE5_HDFE_ENDPOINT), or 3) any point of the cross track is within the longitude-latitude "box" (HE5_HDFE_ANYPOINT), depending on the inclusion mode designated by the user. All elements within an included cross track are considered to be within the region even though a particular element of the cross track might be outside the region. The swath structure must have both <i>Longitude</i> and <i>Latitude</i> (or <i>Colatitude</i>) fields defined |
| Example | In this example, we define a region bounded by the 3 degrees longitude, 5 degrees latitude and 7 degrees longitude, 12 degrees latitude. We will consider a cross track to be within the region if its midpoint is within the region. |

```
cornerlon[0] = 3.;  
cornerlat[0] = 5.;
```

```

cornerlon[1] = 7.;

cornerlat[1] = 12.;

regionID = HE5_SWregionindex(swathID, cornerlon, cornerlat,
                             HE5_HDFE_MIDPOINT, geodim, idxrange);

FORTRAN integer function he5_swregidx(swathid, cornerlon, cornerlat, mode,
                                     geodim, idxrange)
integer          swathid
real*8          cornerlon(*)
real*8          cornerlat(*)
character*(*)
character*(*)
integer*4        idxrange(*)

```

The equivalent *FORTRAN* code for the example above is:

```

parameter (HE5_HDFE_MIDPOINT=0)

cornerlon(1) = 3.

cornerlat(1) = 5.

cornerlon(2) = 7.

cornerlat(2) = 12.

regionid = he5_swregidx(swathid, cornerlon, cornerlat,
                       HE5_HDFE_MIDPOINT, geodim, idxrange)

```

Return Information about a Defined Region

HE5_SWregioninfo

```
herr_t HE5_SWregioninfo(hid_t swathID, hid_t regionID, char *fieldname,  
                        H5T_class_t *ntype, int *rank, hsize_t dims[], size_t *size)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>regionID</i> | IN: Region ID returned by HE5_SWdefboxregion |
| <i>fieldname</i> | IN: Field to subset |
| <i>ntype</i> | OUT: Number type of field |
| <i>rank</i> | OUT: Rank of field |
| <i>dims</i> | OUT: Dimensions of subset region |
| <i>size</i> | OUT: Size in bytes of subset region |
| Purpose | Retrieves information about the subsetted region. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields. |
| Example | In this example, we retrieve information about the region defined in <i>HE5_SWdefboxregion</i> for the <i>Spectra</i> field. We use this to allocate space for data in the subsetted region. <pre>/* Get size in bytes of region for "Spectra" field*/ status = HE5_SWregioninfo(SWid, regionID, "Spectra", &ntype, &rank, dims, &size); /* Allocate space */ datbuf = (double *)calloc(size, sizeof(double));</pre> |

FORTRAN integer function he5_swreginfo(*swathid*, *regionid*, *fieldname*, *ntype*, *rank*,
dims, *size*)

 integer *swathid*
 integer *regionid*
 character*(*) *fieldname*
 integer *ntype*
 integer *rank*
 integer*4 *dims(*)*
 integer*4 *size*

The equivalent *FORTRAN* code for the example above is:

```
status =  
he5_swreginfo(swid,regionid,"Spectra",ntype,rank,dims,size)
```

Create an Alias for Swath Data Field

HE5_SWsetalias

herr_t HE5_SWsetalias(hid_t *swathID*, char **fieldname*, const char **aliaslist*)

| | | |
|------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: | Field name |
| <i>aliaslist</i> | IN: | List of alias(es) to associate with the Data Field |
| Purpose | | Create an alias for Swath data field |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | Creates aliases that can be used to refer to a Swath data field in addition to the name of the field. |
| Example | | In this example, we create an alias for the data field <i>Temperature</i> . <pre>strcpy(aliaslist, "temps 0 to 30"); status = HE5_SWsetalias(swathID, "Temperature", aliaslist);</pre> |
| FORTRAN | | integer function he5_swsetalias (<i>swathid</i> , <i>fieldname</i> , <i>aliaslist</i>) integer <i>swathid</i> character(*) <i>fieldname</i> character(*) <i>aliaslist</i> |

The equivalent *FORTRAN* code for the first example above is:

```
aliaslist = "temps 0 to 30"
status = he5_swsetalias(swathid, "Temperature", aliaslist)
```

Set External Data File(s)

HE5_SWsetextdata

```
herr_t HE5_SWsetextdata(hid_t swathID, const char *filelist, off_t offset[],  
                        hsize_t size[])
```

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>filelist</i> | IN: List of external file names |
| <i>offset[]</i> | IN: Array of offsets (in byte) from the beginning of file to the location in file where the data starts |
| <i>size[]</i> | IN: Array of sizes (in bytes) reserved in the file for the data |
| Purpose | Sets the external data file(s) associated with the data set. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID. |

Example In this example, we set the *ExtData* field:

```
status = HE5_SWsetextdata(swathID, "ext-1.dat,ext-2.dat,ext-  
3.dat", offset, size);
```

FORTRAN integer function he5_swsetxdat(*swathid*,*fllist*,*offset*,*size*)
 integer *swathid*
 integer *status*
 integer*4 *offset(*)*
 integer*4 *size(*)*
 character*(*) *fllist*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swsetxdat(swathid,flist,offset,size)
```

Set Fill Value for a Specified Field

HE5_SWsetfillvalue

```
herr_t HE5_SWsetfillvalue(hid_t swathID, char *fieldname, hid_t ntype, void  
*fillvalue)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Field name |
| <i>ntype</i> | IN: Number type of fill value (should match the number type of a specified field) |
| <i>fillvalue</i> | IN: Pointer to the fill value to be used |
| Purpose | Sets fill value for the specified field. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type. |
| Description | The fill value is placed in all elements of the field which have not been explicitly defined. The field must have 2 or more dimensions. |
| Example | In this example, we set a fill value for the <i>Temperature</i> field: |

```
tempfill = -999.0;  
  
status = HE5_SWsetfillvalue(swathID, "Temperature", ntype,  
&tempfill);
```

FORTRAN integer function he5_swsetfill(*swathid*, *fieldname*, *ntype*, *fillvalue*)
 integer *swathid*
 character(*) *fieldname*
 integer *ntype*
 <valid type> *fillvalue(*)*

The equivalent *FORTRAN* code for the example above is:

```
fillvalue = -999.0  
  
status = he5_swsetfill(swathid, "Temperature", ntype,  
fillvalue)
```

Dismount External Data File

HE5_SWunmount

herr_t HE5_SWunmount(hid_t *swathID*, int *fldgroup*, hid_t *fileID*)

| | | |
|-----------------|---|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fldgroup</i> | IN: | Field group flag |
| <i>fileID</i> | IN: | ID of file returned by HE5_SWmountexternal |
| Purpose | Dismount external data file | |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. | |
| Description | This function dismounts from the external file once the user has completed using the data in the file. | |
| Example | In this example, we dismount from the file used in the previous function <pre>status = HE5_SWunmount(swathID, HE5_HDFE_DATAGROUP, fileID);</pre> | |
| FORTRAN | Not available with this release. | |

Update map index for a specified region

HE5_SWupdateidxmap

```
long HE5_SWupdateidxmap(int swathID, hid_t regionID, long indexin[], long indexout[], long indices[])
```

| | |
|-----------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or Swattach. |
| <i>regionID</i> | IN: Region ID returned by HE5_SWdefboxregion. |
| <i>indexin</i> | IN: The array containing the indices of the data dimension to which each geolocation element corresponds. |
| <i>indexout</i> | OUT: The array containing the indices of the data dimension to which each geolocation corresponds in the subsetted region. The indexout set to NULL, will not be returned. |
| <i>indices</i> | OUT: The array containing the indices for start and stop of region. |
| Purpose | Retrieve indexed array of specified geolocation mapping for a specified region. |
| Return value | Returns size of updated indexed array if successful or FAIL (-1) otherwise. A typical reason for failure is the specified mapping does not exist. |
| Description | This routine retrieves the size of the indexed array and the array of indexed elements of the specified geolocation mapping for the specified region. |
| Example | In this example, we retrieve information about the indexed mapping between the “IdxGeo” and “IdxData” dimensions, defined by HE5_SWdefboxregion: |

```
/* Get size of index_region array */

idxsz = HE5_SWupdateidxmap(swathID, regionID, index, NULL,
                           indices);

/* Allocate memory for index_region */

index_region = (long)malloc(sizeof(long) * idxsz);

/* Get the array index_region */

idxsz = HE5_SWupdateidxmap(swathID, regionID, index,
                           index_region, indices);
```

```
FORTRAN    integer*4 function he5_swupimap(swathid, regionid, indexin, indexout,  
           indices)  
           integer swathid  
           integer regionid  
           integer*4 indexin(*)  
           integer*4 indexout(*)  
           integer*4 indices(2)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_swupdateidxmap(swathid, regionid, index,  
                           index_region, indices)
```

Write/Update Swath Attribute

HE5_SWwriteattr

```
herr_t HE5_SWwriteattr(hid_t swathID, const char *attrname, hid_t ntype,  
                      hsize_t count[], void *datbuf)
```

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | IN: Number type of attribute |
| <i>count</i> | IN: Number of values to store in attribute |
| <i>datbuf</i> | IN: Attribute values |
| Purpose | Writes/Updates attribute in a swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type. |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. |
| Example | In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14: |

```
attr_val = 3.14;  
  
status = HE5_SWwriteattr(swathid, "ScalarFloat",  
                         H5T_NATIVE_FLOAT, 1, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_SWwriteattr(swathid, "ScalarFloat",  
                         H5T_NATIVE_FLOAT, 1, &attr_val);
```

```
FORTRAN    integer function he5_swrrattr(swathid, attrname, ntype, count, datbuf)
            integer      swathid
            character*(*) attrname
            integer*4     count(*)
            <valid type>  datbuf(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT = 1)
datbuf   = 3.14
count    = 1
status = he5_swrrattr(swathid, "ScalarFloat",
HE5_HDFE_NATIVE_FLOAT, count, datbuf)
```

Write Field Metadata for an Existing Swath Data Field

HE5_SWwritedatameta

```
herr_t HE5_SWwritedatameta(hid_t swathID, const char *fieldname, char  
*dimlist, int mvalue)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Name of field |
| <i>dimlist</i> | IN: The list of data dimensions defining the field |
| <i>mvalue</i> | IN: The number type of the data stored in the field |
| Purpose | Writes field metadata for an existing swath data field. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list. |
| Description | This routine writes field metadata for an existing data field. This is useful when the data field was defined without using the swath API. Note that any entries in the dimension list must be defined through the <i>HE5_SWdefdim</i> routine before this routine is called. |
| Example | In this example we write the metadata for the “Band_1” data field used in the swath. |

```
status = HE5_SWwritedatameta(swathID, "Band_1", "GeoTrack,  
GeoXtrack", H5T_NATIVE_FLOAT);
```

FORTRAN

integer function

```
he5_swwrdmeta(swathid,fieldname,dimlist,mvalue)  
integer swathid  
character(*) fieldname  
character(*) dimlist  
integer mvalue
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT = 1)  
status = he5_swwrdmeta(swathID, "Band_1", "GeoXtrack,  
GeoTrack", HE5_HDFE_NATIVE_FLOAT)
```

The dimensions are entered in *FORTRAN* order with the first dimension being incremented first.

Write Data to a Swath Field

HE5_SWwritefield

```
herr_t HE5_SWwritefield(hid_t swathID, char *fieldname, const hsize_t start[],  
                        const hsize_t stride[], const hsize_t edge[], void *data)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Name of field to write |
| <i>start</i> | IN: Array specifying the starting location within each dimension (0-based) |
| <i>stride</i> | IN: Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: Array specifying the number of values to write along each dimension |
| <i>data</i> | IN: Values to be written to the field |
| Purpose | Writes data to a swath field. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or unknown fieldname. |
| Description | The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the swath field (output) dimensions. The input data in the <i>data</i> buffer is read from contiguously. The default values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are $(dim - start) / stride$ where <i>dim</i> refers is the size of the dimension. It is the users responsibility to make sure the data buffer contains sufficient entries to write to the field. Note that the data buffer for a compressed field must be the size of the entire field as incremental writes are not supported by the underlying HDF routines. |
| Example | In this example, we write data to the <i>Longitude</i> field. |

```
float longitude [2000][1000];  
  
/* Define elements of longitude array */  
  
status = HE5_SWwritefield(swathID, "Longitude", NULL, NULL,  
                           NULL, longitude);
```

We now update Track 10 (0 - based) in this field:

```
float newtrack[1000];  
  
hssize_t start[2]={10,0}; hsize_t edge[2]={1,1000};  
  
/* Define elements of newtrack array */  
  
status = HE5_SWwritefield(swathID, "Longitude", start, NULL,  
                           edge, newtrack);
```

FORTRAN integer function

```
he5_swwrfl( swathid,fieldname,start,stride,edge,data)  
  
integer      swathid  
character(*) fieldname  
integer*4    start(*)  
integer*4    stride(*)  
integer*4    edge(*)  
<valid type> data(*)
```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```
real*4      longitude(1000,2000)  
  
integer*4   start(2), stride(2), edge(2)  
  
start(1) = 0  
start(2) = 10  
stride(1) = 1  
stride(2) = 1  
edge(1) = 1000  
edge(2) = 2000  
  
status = he5_swwrfl( swathid, "Longitude", start, stride,  
                     edge, longitude)
```

We now update Track 10 (0 - based) in this field:

```
real*4      newtrack(1000)  
  
integer*4   start(2), stride(2), edge(2)
```

```
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 1000
edge(2) = 1
status = he5_swwrflld(swathid, "Longitude", start, stride,
edge, newtrack)
```

Write Field Metadata to an Existing Swath Geolocation Field

HE5_SWwritegeometa

herr_t HE5_SWwritegeometa(hid_t *swathID*, const char **fieldname*, char **dimlist*, int *mvalue*)

| | |
|------------------|--|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Name of field |
| <i>dimlist</i> | IN: The list of geolocation dimensions defining the field |
| <i>mvalue</i> | IN: The number type of the data stored in the field |
| Purpose | Writes field metadata for an existing swath geolocation field. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reason for failure is unknown dimension in the dimension list. |
| Description | This routine writes field metadata for an existing geolocation field. This is useful when the data field was defined without using the swath API. Note that any entries in the dimension list must be defined through the <i>HE5_SWdefdim</i> routine before this routine is called. |

Example In this example we write the metadata for the *Latitude* geolocation field used in the swath.

```
status = HE5_SWwritegeometa(swathID, "Latitude",
                            "GeoTrack,GeoXtrack",H5T_NATIVE_FLOAT);
```

FORTRAN integer function

```
he5_swwrgmeta(swathid,fieldname,dimlist,mvalue)
integer         swathid
character(*)   fieldname
character(*)   dimlist
integer         mvalue
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT = 1)
```

```
status = he5_swwrgmeta(swathID, "Latitude",
    "GeoXtrack,GeoTrack",HE5_HDFE_NATIVE_FLOAT)
```

The dimensions are entered in *FORTRAN* order with the first dimension being incremented first.

Write/Update Group Swath Attribute

HE5_SWwritegrpattr

```
herr_t HE5_SWwritegrpattr(hid_t swathID, const char *attrname, hid_t ntype,  
                           hsize_t count[], void *datbuf)
```

| | |
|-----------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | IN: Data type of attribute |
| <i>count</i> | IN: Number of values to store in attribute |
| <i>datbuf</i> | IN: Attribute values |
| Purpose | Writes/Updates group attribute in a swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type. |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the "Data Fields" group in the swath file. |

Example In this example, we write a single precision (32 bit) floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_SWwritegrpattr(swathid, "ScalarFloat",  
                           H5T_NATIVE_FLOAT, count, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_SWwritegrpattr(swathid, "ScalarFloat",  
                           H5T_NATIVE_FLOAT, count, &attr_val);
```

```
FORTRAN  integer function he5_swwrgattr(swathid, attrname, ntype, count, dbuf)
          integer      swathid
          character(*) attrname
          integer      ntype
          integer*4    count(*)
          <valid type> dbuf(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)
dbuf = 3.14
count = 1
status = he5_swwrgattr(swathid, "ScalarFloat",
HE5_HDFE_NATIVE_FLOAT,count,dbuf)
```

Write/Update Local Swath Attribute

HE5_SWwritelocattr

```
herr_t HE5_SWwritelocattr(hid_t swathID, const char *fieldname, char  
*attrname, hid_t ntype, hsize_t count[], void *dbuf)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>fieldname</i> | IN: Field name |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | IN: Data type of attribute |
| <i>count</i> | IN: Number of values to store in attribute |
| <i>dbuf</i> | IN: Attribute values |
| Purpose | Writes/Updates group attribute in a swath. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or number type. |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to a particular “Data Field” in the swath file. |
| Example | In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14: |

```
countt[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_SWwritelocattr(swathid, "DataField",  
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_SWwritelocattr(swathid, "DataField",  
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

```
FORTRAN  integer function he5_swwrlattr(swathid, fieldname, attrname, ntype, count,  
          datbuf)  
  
          integer      swathid  
          character(*) fieldname  
          character(*) attrname  
          integer      ntype  
          integer*4    count(*)  
          <valid type> datbuf(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)  
  
datbuf = 3.14  
  
count = 1  
  
status = he5_swwrlattr(swathid, "DataField", "ScalarFloat",  
                      HE5_HDFE_NATIVE_FLOAT, count, datbuf)
```

Define Profile Data Structure

HE5_PRdefine

```
herr_t HE5_PRdefine(hid_t swathID, const char *profilename, chat *dimlist,  
                     char *maxdimlist, hid_t datatype_id)
```

| | | |
|--------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profilename</i> | IN: | Profile name |
| <i>dimlist</i> | IN: | List of profile dimensions (separated by comma) |
| <i>maxdimlist</i> | IN: | List of profile maximum dimensions (separated by comma) |
| <i>dtype</i> | IN: | Base data type ID |
| Purpose | | Sets up a specified profile structure in a swath. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath ID or data type ID. |
| Description | | The profile is linked to the “Data Fields” group in the swath file. |
| Example | | In this example, we define a profile with the name <i>SimpleProfile</i> and with the base ‘unsigned int’ data type. The profile is represented by a single dataset with 4 dimensions. |

```
status = HE5_PRdefine(swathid, "SimpleProfile", dimlist,  
                      maxdimlist, H5T_NATIVE_UINT);
```

FORTRAN

```
integer function he5_prdefine(swathid, profilename, rank, dim,  
                           datatype_id)  
  
    integer          swathid  
    character(*)   profilename  
    character(*)   dimlist  
    character(*)   maxdimlist(*)  
    integer          datatype_id
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_INT = 0)  
  
status = he5_prdefine(swathid, “SimpleProfile”, dimlist, maxdimlist,  
                      HE5_HDFE_NATIVE_INT)
```

Return Information about a Profile in a Swath

HE5_PRinfo

```
herr_t HE5_PRinfo(hid_t swathID, const char *profname, int *rank, hsize_t  
                  dims[], hsize_t maxdims[], H5T_class_t *classID, char  
                  *dimlist, char *maxdimlist)
```

| | | |
|-------------------|------|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profname</i> | IN: | Profile name |
| <i>rank</i> | OUT: | Rank of profile dataset |
| <i>dims</i> | OUT: | Array of dimension sizes |
| <i>maxdims</i> | OUT: | Array of maximum dimension sizes |
| <i>classID</i> | OUT: | Base-data type class ID |
| <i>dimlist</i> | OUT: | Comma separated list of dimension names |
| <i>maxdimlist</i> | OUT: | Comma separated list of maximum dimension names |
| Purpose | | Retrieve information about specified profile dataset in a Swath |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | This routine returns rank, array of dimension and maximum dimension sizes, base data type class ID, comma separated list of dimension and maximum dimension names of profile dataset. |
| Example | | In this example, we retrieve information about profile “Profile-2000”: |

```
status = HE5_PRinfo(swathID,"Profile-2000", rank, dims,  
                     maxdims, classID, dimlist, maxdimlist);
```

FORTRAN

```
integer function he5_prinfo( swathid, profname, rank, dims, maxdims,  
                            classid, dimlist, maxdimlist)  
integer          swathid  
character(*)   profname  
integer         rank  
integer*4       dims(*)  
integer*4       maxdims(*)  
integer         classID  
character(*)   dimlist  
character(*)   maxdimlist
```

The equivalent *FORTRAN* code for the first example above is:

```
profname = "Profile-2000"  
  
status = he5_prinfo(swathid, profname, rank, dims, maxdims,  
classid, dimlist, maxdimlist)
```

Retrieve Information about Profiles in a Swath

HE5_Prinquire

```
long HE5_Prinquire(hid_t swathID, char *profnames, int *rank, H5T_class_t  
*classID)
```

| | |
|------------------|---|
| <i>swathID</i> | IN: Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profnames</i> | OUT: Buffer for returned comma separated list of profile names |
| <i>rank</i> | OUT: Array of ranks of profile datasets |
| <i>classID</i> | OUT: Array of base-data type class IDs of profiles |
| Purpose | Retrieve information about profile datasets in a specified Swath |
| Return value | Returns number of profiles if successful or FAIL (-1) otherwise. |
| Description | A comma separated list of profile datasets is returned. The <i>rank</i> and (base data type) <i>classID</i> arrays will have an entry for each profile. |
| Example | In this example, we retrieve information about profiles: |

```
nprof = HE5_Prinquire(swathID, profnames, rank, classID);
```

FORTRAN

```
integer*4 function he5_prinquire (swathid, profnames, rank, classID)
integer      swathid
character(*) profnames
integer      rank(*)
integer      classID(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
nprof = he5_prinquire(swathid, profnames, rank, classID)
```

Read Data from Profile Structure

HE5_PRread

```
herr_t HE5_PRread(hid_t swathID, const char *profilename, const hsize_t  
                  start[], const hsize_t stride[], const hsize_t edge[], void  
                  *datbuf)
```

| | | |
|--------------------|------|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profilename</i> | IN: | Profile structure name |
| <i>start</i> | IN: | Array specifying starting location within each dimension |
| <i>stride</i> | IN: | Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: | Array specifying the number of values to write along each dimension |
| <i>datbuf</i> | OUT: | Buffer allocated to hold profile values |
| Purpose | | Reads profile data set from a swath. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or incorrect profile name. |
| Description | | After reading the data a call to <i>HE5_PRreclaimspace()</i> should be made to release allocated memory. |
| Example | | In this example, we read an ‘unsigned int’ type profile with the name “SimpleProfile”: |

```
/* Native HDF5 Datatype used in this example: Variable  
Length Datatype struct */  
  
typedef struct {  
  
    size_t len; /* Length of VL data (for base type)*/  
    void *p;    /* Pointer to VL data */  
  
} hvl_t;  
  
  
hvl_t      buffer[4];  
start[0]   = 0;  
stride[0]  = 1;
```

```

edge[ 0 ]      = 4;

status = HE5_PRread(swathID, "SimpleProfile", start, stride,
edge, buffer);

for (i=0; i<4; i++){
    printf("The length of %d-th element is %d \n",
i,(unsigned)buffer[i].len);

    for (j=0; j<2; j++)

        printf("%d \n", ((unsigned int*)buffer[i].p)[j]);

}

status = HE5_PRreclaimspace(swathID, "SimpleProfile",
buffer);

```

FORTRAN

```

integer function
he5_prread(swathid,profname,start,stride,count,len,buffer)

integer          swathid,status
character *(*) profname
integer*4       start(2),stride(2),count(2),len(4)

```

The equivalent *FORTRAN* code for the example above is:

```

start(1) = 0
stride(1) = 1
count(1) = 4

status = he5_prread(swathid, "SimpleProfile", start, stride,
count, len, buffer)

```

Reclaim Memory used by “Read” Buffer

HE5_PRreclaimspace

herr_t HE5_PRreclaimspace(hid_t *swathID*, const char **profilename*, void **buffer*)

| | | |
|--------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profilename</i> | IN: | Profile name |
| <i>buffer</i> | IN: | Data buffer used to read profile dataset |
| Purpose | | Release memory used by the buffer in the call HE5_PRread() |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | | Reclaims memory space allocated to the data buffer in the call <i>HE5_PRread()</i> . |
| Example | | In this example, we reclaim memory allocated for the “read” buffer “buffer” |
| | | <pre>status = HE5_PRreclaimspace(swathID, "Profile-2000", buffer);</pre> |
| FORTRAN | | Not needed. |

Write Data to the Profile Swath Structure

HE5_PRwrite

```
herr_t HE5_PRwrite(int swathID, const char *profilename, const hsize_t start[],  
                    const hsize_t stride[], const hsize_t edge[], size_t size, void  
                    *datbuf)
```

| | | |
|--------------------|-----|---|
| <i>swathID</i> | IN: | Swath ID returned by HE5_SWcreate or HE5_SWattach |
| <i>profilename</i> | IN: | Profile structure name |
| <i>start</i> | IN: | Array specifying the starting location within each dimension (0-based) |
| <i>stride</i> | IN: | Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: | Array specifying the number of values to write along each dimension |
| <i>size</i> | IN: | Size of data buffer (in bytes) for memory allocation routine |
| <i>datbuf</i> | IN: | Profile data values |
| Purpose | | Writes profile data set in a swath. |
| Return value | | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper swath id or profile name. |
| Description | | The specified profile is linked to a "Data Fields" group in the swath file. |
| Example | | In this example, we write data to "SimpleProfile": |

```
size_t          datasize = 0;  
  
hvl_t          buf[4];  
  
for (i = 0; i < 4; i++){  
  
    buf[i].p = malloc(25*(i+1)*sizeof(unsigned int));  
    buf[i].len = 25*(i+1);  
  
    datasize += buf[i].len *sizeof(unsigned int);  
  
    for (j = 0; j < 25*(i+1); j++)  
        ((unsigned int *)buf[i].p)[j] = (i+1)*10+j;  
}
```

```

status = HE5_PRwrite(swathid, "SimpleProfile", start,
stride, edge, datasize, buf);

FORTRAN    integer function he5_prwrite(swathid, profname, start, stride ,count,
datasize, len, buffer)

integer      swathid,status
integer*4     start(3),stride(3),count(3),len(4),datasize
integer      buffer(*), i, j, counter

```

The equivalent *FORTRAN* code for the example above is:

```

datasize = 0
counter = 0
do i=1,4
    len(i) = i*25
    datasize = datasize + len(i)
    do j = 1,(25*i)
        counter = counter + 1
        buffer(counter) = (i)*1000+j-1
    enddo
enddo
start(1) = 0
stride(1) = 1
count(1) = 4
status = he5_prwrite(swathid, "SimplePrifile", start,
stride, count, datasize, len, buffer)

```

2.1.3 Grid Interface Functions

This section contains an alphabetical listing of all the functions in the Grid interface. The functions are alphabetized based on their C-language names.

Attach to an Existing Grid Structure

HE5_GDattach

hid_t HE5_GDattach(hid_t *fid*, char **gridname*)

| | |
|-----------------|---|
| <i>fid</i> | IN: Grid file ID returned by HE5_GDopen |
| <i>gridname</i> | IN: Name of grid to be attached |
| Purpose | Attaches to an existing grid within the file. |
| Return value | Returns the grid handle(<i>gridID</i>) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper grid file id or grid name. |
| Description | This routine attaches to the grid using the <i>gridname</i> parameter as the identifier. |
| Example | In this example, we attach to the previously created grid, " <i>ExampleGrid</i> ", within the HDF-EOS file, <i>Grid.he5</i> , referred to by the handle, <i>fid</i> : |

```
gridID = HE5_GDattach(fid, "ExampleGrid");
```

The grid can then be referenced by subsequent routines using the handle, *gridID*.

FORTRAN integer function he5_gdattach(*fid*, *gridname*)
 integer *fid*
 character*(*) *gridname*

The equivalent *FORTRAN* code for the example above is:

```
gridid = he5_gdattach(fid, "ExampleGrid")
```

Return Information about a Grid Attribute

HE5_GDattrinfo

```
herr_t HE5_GDattrinfo(hid_t gridID, const char *attrname, H5T_class_t * ntype,  
                      hsize_t *count)
```

| | |
|-----------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | OUT: Number type of attribute |
| <i>count</i> | OUT: Number of total bytes in attribute |
| Purpose | Returns information about a grid attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a grid attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_GDattrinfo(gridID,"ScalarFloat",&ntype,&count);
```

The *ntype* variable will have the value 5 and *count* will have the value 4.

FORTRAN

```
integer function he5_gdattrinfo(gridid, attrname, ntype, count,)  
integer      gridid  
character(*) attrname  
integer      ntype  
integer*4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_gdattrinfo(gridid, "ScalarFloat", ntype, count)
```

Write Block SOM Offset

HE5_GDblkSOMOffset

herr_t HE5_GDblkSOMOffset(hid_t *gridID*, long *offset*[], hsize_t *count*, char **code*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

offset IN: Offset values for SOM Projection data

count IN: Number of offset values to write

code IN: Write/Read code

Purpose Write block SOM offset values.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description The routine supports structures that contain data which have been written in the Solar Oblique Mercator (SOM) projection. The structure can contain one to many blocks, each with corner points defined by latitude and longitude. The routine can only be used by grids that use the SOM projection. The routine writes the offset values, in pixels, from a standard SOM projection. There is an offset value for every block in the grid except for the first block. The count parameter is used as a check for the number of offset values. This routine will also return the offset values, but the user must know how large the offset array needs to be before calling the function, in that case the code value would be "r" and the count parameter has to be provided also.

Example In this example, we first show how the SOM projection is defined using HE5_GDdefproj, then we show how the SOM projection is modified using HE5_GDblkSOMOffset:

The first parameter is the Grid ID, the second is the projection code for the SOM projection, the third is the zone code, not needed for the SOM projection, the fourth is the sphere code, not needed for the SOM projection and the last parameter is the projection parameter array. Each projection supported by the Grid interface has a unique set of variables that are used by the GCTP library and they are passed to the GCTP library through this array. As you can see below, the twelfth parameter is set to a non-zero value, it is set to the size of the number of blocks in the data field. This is required if the function HE5_GDblkSOMOffset is going to be called. The GCTP library doesn't use the this parameter for the SOM projection so that is used by the HDF-EOS library only. The

HE5_GDblkSOMoffset function checks that parameter first before anything else is done.

```
projparm[0] = 6378137.0;  
projparm[1] = 0.006694348;  
projparm[3] = HE5_EHconvAng(98.161, HE5_HDFE_DEG_DMS);  
projparm[4] =  
    HE5_EHconvAng(87.11516945924,HE5_HDFE_DEG_DMS);  
projparm[8] = 0.068585416 * 1440;  
projparm[9] = 0.0;  
projparm[11] = 6;  
status = HE5_GDdefproj(GDid_som, HE5_GCTP_SOM, NULL, NULL,  
projparm);
```

Now that the projection has been defined, HE5_GDblkSOMoffset can be called:

```
offset[5] = {5, 10, 12, 8, 2};  
count = 5;  
code = "w";  
status = HE5_GDblkSOMoffset(gridID, offset, count, code);
```

This set the offset for the second block to 5 pixels, the third block to 10 pixels, fourth block to 12 pixels, fifth to 8 pixels and the sixth block to 2 pixels.

NOTE: This routine is currently implemented in “C” only. If the need arises, a FORTRAN function will be added.

Interblock subsetting is not currently supported by the ECS Science Data Server, at this time. That is, a response to a request to return data contained within a specified latitude/longitude box, will be in an integral number of blocks.

Related Documents

An Album of Map Projections, USGS Professional Paper 1453, Snyder and Voxland, 1989

Map Projections - A Working Manual, USGS Professional Paper 1395, Snyder, 1987

Close an HDF-EOS File

HE5_GDclose

herr_t HE5_GDclose(hid_t *fid*)

fid IN: Grid file ID returned by HE5_GDopen

Purpose Closes file.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine closes the HDF-EOS grid file.

Example

```
status = HE5_GDclose(fid);
```

FORTRAN integer function he5_gdclose(*fid*)
integer *fid*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdclose(fid)
```

Retreive Compression Information for Field

HE5_GDcompinfo

```
herr_t HE5_GDcompinfo(hid_t gridID, const char *fieldname, char *compcode,  
                      int compparm[])
```

| | |
|------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Fieldname |
| <i>compcode</i> | OUT: HDF compression code |
| <i>compparm</i> | OUT: Compression parameters |
| Purpose | Retrieves compression information about a field. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. |
| Description | This routine returns the compression code and compression parameters for a given field. |
| Example | To retreive the compression information about the <i>Opacity</i> field defined in the <i>HE5_GDdefcomp</i> section: |

```
status = HE5_GDcompinfo(gridID, "Opacity", compcode,  
                      compparm);
```

The *compcode* parameter will be set to 4 and *compparm*[0] to 5.

FORTRAN

```
integer function he5_gdcompinfo(gridid,fieldname compcode, compparm)  
    integer      gridid  
    character(*) fieldname  
    character(*) compcode  
    integer      compparm(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdcompinfo(gridid, 'Opacity', compcode,  
                      compparm)
```

The *compcode* parameter will be set to 4 and *compparm*(1) to 5.

Create a New Grid Structure

HE5_GDcreate

```
hid_t HE5_GDcreate(hid_t fid, const char *gridname, long xdimsize, long  
                  ydimsize, double upleftpt[], double lowrightpt[])
```

| | |
|-------------------|--|
| <i>fid</i> | IN: Grid file ID returned by HE5_GDopen |
| <i>gridname</i> | IN: Name of grid to be created |
| <i>xdimsize</i> | IN: Number of columns in grid |
| <i>ydimsize</i> | IN: Number of rows in grid |
| <i>upleftpt</i> | IN: Location, of upper left corner of the upper left pixel |
| <i>lowrightpt</i> | IN: Location, of lower right corner of the lower right pixel |
| Purpose | Creates a grid within the file. |
| Return value | Returns the grid handle(<i>gridID</i>) or FAIL(-1) otherwise. |
| Description | The grid is created as a group within the HDF-EOS file with the name <i>gridname</i> . This routine establishes the resolution of the grid, ie, the number of rows and columns, and it's location within the complete global projection through the <i>upleftpt</i> and <i>lowrightpt</i> arrays. These arrays should be in meters for all GCTP projections other than the Geographic Projection, which should be in packed degree format. q.v. below. |
| Example | In this example, we create a UTM grid bounded by 54 E to 60 E longitude and 20 N to 30 N latitude. We divide it into 120 bins along the x-axis and 200 bins along the y-axis |

```
uplft[0]=210584.50041;  
uplft[1]=3322395.95445;  
lowrgt[0]=813931.10959;  
lowrgt[1]=2214162.53278;  
xdim=120;  
ydim=200;  
gridID = HE5_GDcreate(fid, "UTMGrid", xdim, ydim, uplft,  
                      lowrgt);
```

The grid structure is then referenced by subsequent routines using the handle, *gridID*.

The *xdim* and *ydim* values are referenced in the field definition routines by the reserved dimensions: *XDim* and *YDim*.

For the Polar Stereographic, Goode Homolosine and Lambert Azimuthal projections, we have established default values in the case of an entire hemisphere for the first projection, the entire globe for the second and the entire polar or equitorial projection for the third. Thus, if we have a Polar Stereographic projection of the Northern Hemisphere then the *uplft* and *lowrgt* arrays can be replaced by *NULL* in the function call.

In the case of the Geographic projection (linear scale in both longitude latitude), the *upleftpt* and *lowrightpt* arrays contain the longitude and latitude of these points in packed degree format (DDDMMMSSS.SS).

Note:

upleftpt - Array that contains the X-Y coordinates of the upper left corner of the upper left pixel of the grid. First and second elements of the array contain the X and Y coordinates respectively. The upper left X coordinate value should be the lowest X value of the grid. The upper left Y coordinate value should be the highest Y value of the grid.

lowrightpt - Array that contains the X-Y coordinates of the lower right corner of the lower right pixel of the grid. First and second elements of the array contain the X and Y coordinates respectively. The lower right X coordinate value should be the highest X value of the grid. The lower right Y coordinate value should be the lowest Y value of the grid.

If the projection id geographic (i.e., projcode=0) then the X-Y coordinates should be specified in degrees/minutes/seconds (DDDMMMSSS.SS) format. The first element of the array holds the longitude and the second element holds the latitude. Latitudes are from -90 to +90 and longitudes are from -180 to +180 (west is negative).

For all other projection types the X-Y coordinates should be in **meters** in double precision. These coordinates have to be computed using the GCTP software with the same projection parameters that have been specified in the projparm array. For UTM projections use the same zone code and its sign (positive or negative) while computing both upper left and lower right corner X-Y coordinates irrespective of the hemisphere.

To convert lat/long to x-y coordinates, it is also possible to use SDP Toolkit routines: PGS_GCT_Init() or PGS_GCT_Proj(). More information is contained in the *SDP Toolkit Users Guide for the ECS Project*

```
FORTRAN    integer function he5_gdcreate(fid, gridname, xdimsize, ydimsize, upleftpt,  
           lowrightpt)  
           integer      fid  
           character(*) gridname  
           integer*4    xdimsize  
           integer*4    ydimsize  
           real*8      upleftpt(2)  
           real*8      lowrightpt(2)
```

The equivalent *FORTRAN* code for the example above is:

```
gridid = he5_gdcreate(fid, "UTMGrid", xdim, ydim, uplft,  
                      lowrgt)
```

The default values for the Polar Stereographic and Goode Homolosine can be designated by setting all elements in the *uplft* and *lowrgt* arrays to 0.

Define Region of interest by Latitude/Longitude

HE5_GDdefboxregion

hid_t HE5_GDdefboxregion(hid_t *gridID*, double *cornerlon[]*, double *cornerlat[]*)

| | |
|------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>cornerlon</i> | IN: Longitude in decimal degrees of box corners |
| <i>cornerlat</i> | IN: Latitude in decimal degrees of box corners |
| Purpose | Defines a longitude-latitude box region for a grid. |
| Return value | Returns the grid region ID if successful or FAIL (-1) otherwise. |
| Description | This routine defines a longitude-latitude box region for a grid. It returns a grid region ID which is used by the <i>HE5_GDextractregion</i> routine to read all the entries of a data field within the region. |

Example In this example, we define the region to be the first quadrant of the Northern hemisphere.

```
cornerlon[0] = 0.;      cornerlat[0] = 90.;  
cornerlon[1] = 90.;     cornerlat[1] = 0.;  
regionID = HE5_GDdefboxregion(GDid, cornerlon, cornerlat);
```

FORTRAN integer function he5_gddefboxreg(*gridid*, *cornerlon*, *cornerlat*)
integer *gridid*
real*8 *cornerlon*(2)
real*8 *cornerlat*(2)

The equivalent *FORTRAN* code for the example above is:

```
cornerlon(1) = 0.  
cornerlat(1) = 90.  
cornerlon(2) = 90.  
cornerlat(2) = 0.  
regionid = he5_gddefboxreg(gridid, cornerlon,cornerlat)
```

Set Grid Field Compression

HE5_GDdefcomp

herr_t HE5_GDdefcomp(hid_t *gridID*, int *comppcode*, int *comppparm[]*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

comppcode IN: HDF compression code

comppparm IN: Compression parameters (if applicable)

Note: Only deflate compression is supported in this release.

Purpose Sets the field compression for all subsequent field definitions.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine sets the HDF field compression for subsequent grid field definitions. The compression does not apply to one-dimensional fields. The compression schemes currently supported are: run length encoding (HE5_COMP_DEFLATE=4) and no compression (HE5_COMP_NONE = 0, the default). Deflate compression requires a single integer compression parameter in the range of one to nine with higher values corresponding to greater compression. Compressed fields are written using the standard *HE5_GDwritefield* routine, however, the entire field must be written in a single call. If this is not possible, the user should consider tiling. See *HE5_GDdeftile* for further information. Any portion of a compressed field can then be accessed with the *HE5_GDreadfield* routine. Compression takes precedence over merging so that multi-dimensional fields that are compressed are not merged. The user should refer to the HDF Reference Manual for a fuller explanation of the compression schemes and parameters.

Example Suppose we wish to compress the fields *Pressure*, *Opacity* and *Spectra* using deflate compression, and use no compression for the *Temperature* field.

```
status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_DEFLATE, NULL);  
  
status = HE5_GDdeffield(gridID, "Pressure",  
    "YDim,XDim",NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);  
  
status = HE5_GDdeffield(gridID, "Opacity", "YDim,XDim",  
    NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);
```

```

status = HE5_GDdeffield(gridID, "Spectra", "Bands,YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

status = HE5_GDdefcomp(gridID, HE5_HDFE_COMP_NONE, NULL);

status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim",
NULL, H5T_NATIVE_FLOAT, HE5_HDFE_NOMERGE);

```

Note that the HE5_HDFE_NOMERGE parameter will be ignored in the field definitions.

FORTRAN

```

integer function he5_gddefcomp(gridid, compcode, compparm)
integer      gridid
integer      compcode
integer      compparm(*)

```

The equivalent *FORTRAN* code for the example above is:

```

parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_COMP_NONE=0)
parameter (HE5_HDFE_COMP_DEFLATE=4)
parameter (HE5_HDFE_NOMERGE = 0)

status = he5_gddefcomp(gridid, HE5_HDFE_COMP_DEFLATE,
compparm)
status = he5_gddeffld(gridid, "Pressure", "YDim,XDim", " ",
HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
status = he5_gddeffld(gridid, "Opacity", "YDim,XDim",
" ", HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
status = he5_gddeffld(gridid, "Spectra", "Bands,YDim,XDim",
" ", HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
status = he5_gddefcomp(gridid, HE5_HDFE_COMP_NONE, compparm)
status = he5_gddeffld(gridid, "Temperature", "YDim,XDim",
" ", HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)

```

Define Compression with Data Tiling

HE5_GDdefcomtile

`herr_t HE5_GDdefcomtile(hid_t gridID, int compcode, int *compparm, int tilerank, const hsize *tiledim)`

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

compcode IN: Compression method flag

compparm IN: Array of compression parameters

tilerank IN: Rank of a field to compress

tiledim IN: Array of sizes of tile

Purpose Compress the data field

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise.

Description This function allows the user to set compression for a data field with automatic tiling

Example In this example, we set (DEFLATE) compression for a field that is defined right after this call

```
compcode = 4;  
compparm[0] = 6;  
  
status = HE5_GDdefcomtile(gridID, compcode, compparm,  
tilerank, tiledim);
```

FORTRAN integer function he5_gddefcomtle(gridid, compcode, compparm, tilerank, tiledim)

integer *gridid*

integer *compcode*

integer *compparm(*)*

integer *tilerank*

integer*4 *tiledim*

The equivalent *FORTRAN* code for the example above is

```
status = he5_gddefcomtle(gridid, compcode, compparm,tilerank,tiledim)
```

Define a New Dimension within a Grid

HE5_GDdefdim

herr_t HE5_GDdefdim(hid_t *gridID*, char **dimname*, hsize_t *dim*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

dimname IN: Name of dimension to be defined

dim IN: The size of the dimension

Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”

Purpose Defines a new dimension within the grid.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an improper grid id.

Description This routine defines dimensions that are used by the field definition routines (described subsequently) to establish the size of the field.

Example In this example, we define a dimension, *Band*, with size 15.

```
status = HE5_GDdefdim(gridID, "Band", 15);
```

To specify an unlimited dimension which can be used to define an appendable array, the dimension value should be set to zero or equivalently, *H5S_UNLIMITED*:

```
status = HE5_GDdefdim(gridID, "Unlim", H5S_UNLIMITED);
```

FORTRAN integer function he5_gddefdim(*gridid*, *fieldname*, *dim*)

integer *gridid*

character(*) *fieldname*

integer*4 *dim*

The equivalent *FORTRAN* code for the example above is:

```
parameter (H5S_UNLIMITED=-1)
```

```
dim = 15
```

```
status = he5_gddefdim(gridid, "Band", dim)
```

```
status = he5_gddefdim(gridid, "Unlim", H5S_UNLIMITED)
```

Define a New Data Field within a Grid

HE5_GDdeffield

```
herr_t HE5_GDdeffield(hid_t gridID, const char *fieldname, char *dimlist, char  
*maxdimlist hid_t ntype, int merge)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

fieldname IN: Name of field to be defined

dimlist IN: The list of data dimensions defining the field

maxdimlist IN: The maximum dimensions list defining the field

ntype IN: The number type of the data stored in the field

merge IN: Merge code (HE5_HDFE-NOMERGE (0) - no merge,
HE5_HDFE_AUTOMERGE (1) -merge)

Note: Merging is not supported in this release of the library. There are three illegal characters for field names: “/”, “;”, “,”

Purpose Defines a new data field within the grid.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reason for failure is an unknown dimension in the dimension list.

Description This routine defines data fields to be stored in the grid. The dimensions are entered as a string consisting of geolocation dimensions separated by commas. They are entered in C order, that is, the last dimension is incremented first.

Example In this example, we define a grid field, *Temperature* with dimensions *XDim* and *YDim* (as established by the *HE5_GDcreate* routine) containing 4-byte floating point numbers and a field, *Spectra*, with dimensions *XDim*, *YDim*, and *Bands*:

```
status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim",  
NULL, H5T_NATIVE_FLOAT, 0);  
  
status = HE5_GDdeffield(gridID, "Spectra",  
"Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT, 0);
```

FORTRAN integer function he5_gddeffld(*gridid*, *fieldname*, *dimlist*, *maxdimlist*,
ntype, *merge*)

integer *gridid*

```
character*(*) fieldname
character*(*) dimlist
character*(*) maxdimlist
integer      ntype, merge
```

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_NOMERGE=0)

status = he5_gddeffld(gridid, "Temperature", "XDim,YDim", "
", HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)

status = he5_gddeffld(gridid, "Spectra", "XDim,YDim,Bands",
" ", HE5_HDFE_NATIVE_FLOAT,HE5_HDFE_NOMERGE)
```

The dimensions are entered in *FORTRAN* order with the first dimension incremented first.

Define the Origin of the Grid Data

HE5_GDdeforigin

herr_t HE5_GDdeforigin(hid_t *gridID*, int *origincode*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

origincode IN: Location of the origin of the grid data

Purpose Defines the origin of the grid data

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description The routine is used to define the origin of the grid data. This allows the user to select any corner of the grid as the origin.

Origin Codes:

HE5_HDFE_GD_UL(Default) (0) Upper Left corner of grid

HE5_HDFE_GD_UR (1) Upper Right corner of grid

HE5_HDFE_GD_LL (2) Lower Left corner of grid

HE5_HDFE_GD_LR (3) Lower Right corner of grid

Example In this example we define the origin of the grid to be the Lower Right corner:

```
status = HE5_GDdeforigin(gridID, HE5_HDFE_GD_LR);
```

FORTRAN integer function he5_gddeforg(*gridid*, *origincode*)

integer *gridid*

integer *origincode*

The equivalent *FORTRAN* code for the above example is :

```
parameter (HE5_HDFE_GD_LR=3)
```

```
status = he5_gddeforg(gridid, HE5_HDFE_GD_LR)
```

Define a Pixel Registration within a Grid

HE5_GDdefpixreg

herr_t HE5_GDdefpixreg(hid_t *gridID*, int *pixregcode*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

pixregcode IN: Pixel registration code

Purpose Defines pixel registration within grid cell

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine is used to define whether the pixel center or pixel corner (as defined by the *HE5_GDdeforigin* routine) is used when requesting the location (longitude and latitude) of a given pixel.

Registration Codes:

HE5_HDFE_CENTER (0) (Default) Center of pixel cell

HE5_HDFE_CORNER (1) Corner of a pixel cell

Example In this example, we define the pixel registration to be the corner of the pixel cell:

```
status = HE5_GDdefpixreg(gridID, HE5_HDFE_CORNER);
```

FORTRAN integer function he5_gddefpixreg(*gridid*, *pixregcode*)

integer *gridid*

integer *pixregcode*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_CORNER=1)
```

```
status = he5_gddefpixreg(gridid, HE5_HDFE_CORNER)
```

Define Grid Projection

HE5_GDdefproj

```
herr_t HE5_GDdefproj(hid_t gridID, int projcode, int zonecode, int spherocode,  
                      double projparm[])
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>projcode</i> | IN: GCTP projection code |
| <i>zonecode</i> | IN: GCTP zone code used by UTM projection |
| <i>spherocode</i> | IN: GCTP spheroid code |
| <i>projparm</i> | IN: GCTP projection parameter array |
| Purpose | Defines projection of grid |
| Return Value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise |
| Description | Defines the GCTP projection and projection parameters of the grid. |
| Example | In this example, we define a Universal Transverse Mercator (UTM) grid bounded by 54 E - 60 E longitude and 20 N - 30 N latitude – UTM zonecode 40, using default spheroid (Clarke 1866), spherocode = 0 |

```
spherocode = 0;  
zonecode = 40;  
  
status = HE5_GDdefproj(gridID, HE5_GCTP_UTM, zonecode,  
                      spherocode, NULL);
```

In this next example we define a Polar Stereographic projection of the Northern Hemisphere (True scale at 90 N, 0 Longitude below pole) using the International 1967 spheroid.

```
spherocode = 3;  
  
for (i = 0; i < 13; i++) projparm[i] = 0;  
  
/* Set Long below pole & true scale in DDDMMSSS.SSS form */  
projparm[5] = 90000000.00;  
  
status = HE5_GDdefproj(gridID, HE5_GCTP_PS, NULL,  
                      spherocode, projparm);
```

Finally we define a Geographic projection. In this case neither the zone code, sphere code or the projection parameters are used.

```

status = HE5_GDdefproj(gridID, HE5_GCTP_GEO, NULL, NULL,
NULL)

FORTRAN    integer function he5_gddefproj(gridid, projcode, zonecode, spherecode,
                                         projparm)

            integer      gridid
            integer      projcode
            integer      zonecode
            integer      spherecode
            real*8       projparm(*)

```

The equivalent FORTRAN code for the examples above is:

```

parameter (HE5_GCTP_UTM=1)

spherecode = 0

zonecode = 40

status = he5_gddefproj(gridid, HE5_GCTP_UTM, zonecode,
spherecode, dummy)

parameter (HE5_GCTP_PS=6)

spherecode = 6

do i=1,13

    projparm(i) = 0

enddo

projparm(6) = 90000000.00

status = he5_gddefproj(gridid, HE5_GCTP_PS, dummy,
spherecode, projparm)

parameter (GCTP_GEO=0)

status = he5_gddefproj(gridid, HE5_GCTP_GEO, dummy, dummy,
dummy)

```

Note: projcode, zonecode, spherecode and projection parameter information are listed in Section 1.6, GCTP Usage.

Define Tiling Parameters

HE5_GDdeftile

```
herr_t HE5_GDdeftile(hid_t gridID, int tilecode, int tilerank, const hsize_t *tiledims[])
```

| | | |
|-----------------|-----|---|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>tilecode</i> | IN: | Tile code: HE5_HDF_TILE, HE5_HDF_NOTILE (default) |
| <i>tilerank</i> | IN: | The number of tile dimensions |
| <i>tiledims</i> | IN: | Tile dimensions |
| Purpose | | Defines tiling dimensions for subsequent field definitions |
| Return Value | | Returns SUCCEED(0) if successful or FAIL(-1) otherwise |
| Description | | This routine defines the tiling dimensions for fields defined following this function call, analogous to the procedure for setting the field compression scheme using <i>HE5_GDdefcomp</i> . The number of tile dimensions and subsequent field dimensions must be the same and the tile dimensions must be integral divisors of the corresponding field dimensions. A tile dimension set to 0 will be equivalent to 1. |

| | |
|---------|--|
| Example | We will define four fields in a grid, two two-dimensional fields of the same size with the same tiling, a three-dimensional field with a different tiling scheme, and a fourth with no tiling. We assume that <i>XDim</i> is 200 and <i>YDim</i> is 300. |
|---------|--|

```
tiledims[0] = 100;  
tiledims[1] = 200;  
  
status = HE5_GDdeftile(gridID, HE5_HDFE_TILE, 2, tiledims);  
  
status = HE5_GDdeffield(gridID, "Pressure", "YDim,XDim",  
NULL, H5T_NATIVE_INT, 0);  
  
status = HE5_GDdeffield(gridID, "Temperature", "YDim,XDim",  
NULL, H5T_NATIVE_FLOAT, 0);  
  
tiledims[0] = 1;  
tiledims[1] = 150;  
tiledims[2] = 100;  
  
status = HE5_GDdeftile(gridID, HE5_HDFE_TILE, 3, tiledims);
```

```

status = HE5_GDdeffield(gridID, "Spectra",
    "Bands,YDim,XDim", NULL, H5T_NATIVE_FLOAT,
    HE5_HDFE_NOMERGE);

status = HE5_GDdeftile(gridID, HE5_HDFE_NOTILE, 0, NULL);

status = HE5_GDdeffield(gridID, "Communities", "YDim,XDim",
    NULL, H5T_NATIVE_INT, HE5_HDFE_AUTOMERGE);

```

FORTRAN integer function he5_gddeftle(*gridid*, *tilecode*,*tilerank*,*tiledims*)
 integer *gridid*
 integer *tilecode*
 integer *tilerank*
 integer*4 *tiledims*(*)

The equivalent *FORTRAN* code for the example above is:

```

parameter (HE5_HDFE_NATIVE_INT=0)
parameter (HE5_HDFE_NATIVE_FLOAT=1)
parameter (HE5_HDFE_NOTILE=0)
parameter (HE5_HDFE_TILE=1)
parameter (HE5_HDFE_NOMERGE = 0)
tiledims(1) = 200
tiledims(2) = 100
tilerank     = 2
status = he5_gddeftle(gridid, HE5_HDFE_TILE,tilerank,
tiledims)

status = he5_gddeffld(gridid, 'Pressure', 'XDim,YDim', " ",
HE5_HDFE_NATIVE_INT, HE5_HDFE_NOMERGE)

status = he5_gddeffld(gridid, 'Temperature', 'XDim,YDim',
" ", HE5_HDFE_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

tiledims[1] = 100
tiledims[2] = 150
tiledims[3] = 1
tilerank     = 3

```

```
status = he5_gddeftle(gridid, HE5_HDFE_TILE, tilerank,
tiledims)

status = he5_gddeffld(gridid, 'Spectra', 'XDim,YDim,Bands',
" ", HE5_HDFE_NATIVE_FLOAT, HE5_HDFE_NOMERGE)

tilerank = 2

status = he5_gddeftle(gridid, HE5_HDFE_NOTILE, tilerank,
tiledims);

status = he5_gddeffld(gridid, 'Communities', 'XDim,YDim',
" ", HE5_HDFE_NATIVE_INT, HE5_HDFE_AUTOMERGE)
```

Define a Time Period of interest

HE5_GDdeftimeperiod

```
herr_t HE5_GDdeftimeperiod(hid_t gridID, hid_t periodID, double starttime,  
                           double stoptime)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

periodID IN: Period (or region) id from previous subset call

starttime IN: Start time of period

stoptime IN: Stop time of period

Purpose Defines a time period for a grid.

Return value Returns the grid period ID if successful or FAIL (-1) otherwise.

Description This routine defines a time period for a grid. It returns a grid period ID which is used by the *HE5_GDextractperiod* routine to read all the entries of a data field within the time period.. The grid structure must have the *Time* field defined. This routine may be called after *HE5_GDdefboxregion* to provide both geographic and time subsetting . In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) Furthermore it can be called before or after *HE5_GDdefvrregion* to further refine a region. This routine may also be called “stand-alone” by setting the input id to HE5_HDFE_NOPREVSUB (-1).

Example In this example, we define a time period with a start time of 35232487.2 and a stop time of 36609898.1.

```
starttime = 35232487.2;  
stoptime = 36609898.1;  
periodID = HE5_GDdeftimeperiod(gridID, HE5_HDFE_NOPREVSUB  
                                 starttime, stoptime);
```

If we had previously performed a geographic subset with id, *regionID*, then we could further time subset this region with the call:

```
periodID = HE5_GDdeftimeperiod(gridID, regionID, starttime,  
                                 stoptime);
```

Note that *periodID* will have the same value as *regionID*.

FORTRAN

```
integer function he5_gddeftmeper(gridid, periodID, starttime, stoptime)
integer      gridid
integer      periodid
real*8       starttime
real*8       stoptime
```

The equivalent *FORTRAN* code for the examples above are:

```
parameter (HE5_HDFE_NOPREVSUB=-1)
starttime = 35232487.2
stoptime  = 36609898.1
periodid = he5_gddeftmeper(gridid, HE5_HDFE_NOPREVSUB,
                           starttime, stoptime)

periodid = he5_gddeftmeper(gridid, regionid, starttime,
                           stoptime)
```

Define a Vertical Subset Region

HE5_GDdefvrtrregion

```
hid_t HE5_GDdefvrtrregion(hid_t gridID, hid_t regionID, char *vertObj, double range[])
```

| | |
|-----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>regionID</i> | IN: Region (or period) id from previous subset call |
| <i>vertObj</i> | IN: Dimension or field to subset |
| <i>range</i> | IN: Minimum and maximum range for subset |
| Purpose | Subsets on a monotonic field or contiguous elements of a dimension. |
| Return value | Returns the grid region ID if successful or FAIL (-1) otherwise. |
| Description | Whereas the <i>HE5_GDdefboxregion</i> routine subsets along the <i>XDim</i> and <i>YDim</i> dimensions, this routine allows the user to subset along any other dimension. The region is specified by a set of minimum and maximum values and can represent either a dimension index (case 1) or field value range(case 2). In the second case, the field must be one-dimensional and the values must be monotonic (strictly increasing or decreasing) in order that the resulting dimension index range be contiguous. (For the current version of this routine, the second option is restricted to fields with number type: INT, LONG, FLOAT, DOUBLE.) This routine may be called after <i>HE5_GDdefboxregion</i> to provide both geographic and “vertical” subsetting. In this case the user provides the id from the previous subset call. (This same id is then returned by the function.) This routine may also be called “stand-alone” by setting the input id to HE5_HDFE_NOPREVSUB (-1). |
| | This routine may be called up to eight times with the same region ID. It this way a region can be subsetted along a number of dimensions. |
| | The <i>HE5_GDregioninfo</i> and <i>HE5_GDextractregion</i> routines work as before, however the field to be subsetted, (the field specified in the call to <i>HE5_GDregioninfo</i> and <i>HE5_GDextractregion</i>) must contain the dimension used explicitly in the call to <i>HE5_GDdefvrtrregion</i> (case 1) or the dimension of the one-dimensional field (case 2). |
| Example | Suppose we have a field called <i>Pressure</i> of dimension <i>Height</i> (= 10) whose values increase from 100 to 1000. If we desire all the elements with values between 500 and 800, we make the call: |

```

range[0] = 500. ;
range[1] = 800. ;
regionID = HE5_GDdefvrtrregion(gridID, HE5_HDFE_NOPREVSUB,
"Pressure", range);

```

The routine determines the elements in the *Height* dimension which correspond to the values of the *Pressure* field between 500 and 800.

If we wish to specify the subset as elements 2 through 5 (0 - based) of the *Height* dimension, the call would be:

```

range[0] = 2;
range[1] = 5;
regionID = HE5_GDdefvrtrregion(gridID, HE5_HDFE_NOPREVSUB,
"DIM:Height", range);

```

The “DIM:” prefix tells the routine that the range corresponds to elements of a dimension rather than values of a field.

If a previous subset region or period was defined with id, *subsetID*, that we wish to refine further with the vertical subsetting defined above we make the call:

```
regionID = HE5_GDdefvrtrregion(gridID, subsetID, "Pressure",
range);
```

The return value, *regionID* is set equal to *subsetID*. That is, the subset region is modified rather than a new one created.

In this example, any field to be subsetted must contain the *Height* dimension.

FORTRAN

```

integer function he5_gddefvrtrreg(gridid, regionid, vertobj, range)
integer      gridid
integer      regionid
character(*) vertobj
real*8       range(2)

```

The equivalent *FORTRAN* code for the examples above is:

```

parameter (HE5_HDFE_NOPREVSUB=-1)
range(1) = 500.
range(2) = 800.
regionid = he5_gddefvrtrreg(gridid, HE5_HDFE_NOPREVSUB,
"Pressure", range)
range(1) = 3          ! Note 1-based element numbers
range(2) = 6

```

```
regionid = he5_gddefvrtrg(gridid, HE5_HDFE_NOPREVSUB,  
"DIM:Height", range)  
regionid = he5_gddefvrtrg(gridid, subsetid, "Pressure",  
range)
```

Detach from Grid Structure

HE5_GDdetach

herr_t HE5_GDdetach(hid_t *gridID*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

Purpose Detaches from grid interface.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine should be run before exiting from the grid file for every grid opened by *HE5_GDcreate* or *HE5_GDattach*.

Example In this example, we detach the grid structure, *ExampleGrid*:

```
status = HE5_GDdetach(gridID);
```

FORTRAN integer function he5_gddetach(gridid)

```
integer      gridid
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gddetach(gridid)
```

Retrieve Size of Specified Dimension

HE5_GDdiminfo

hsize_t HE5_GDdiminfo(hid_t *gridID*, char **dimname*)

| | |
|----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>dimname</i> | IN: Dimension name |
| Purpose | Retrieve size of specified dimension. |
| Return value | Size of dimension if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id or dimension name. |
| Description | This routine retrieves the size of specified dimension. |

Example In this example, we retrieve information about the dimension, "Bands":

```
dimsize = HE5_GDdiminfo(gridID, "Bands");
```

The return value, *dimsize*, will be equal to 15

FORTRAN integer*4 function he5_gddiminfo(*gridid*,*dimname*)
integer *gridid*
character(*) *dimname*

The equivalent *FORTRAN* code for the example above is:

```
dimsize = he5_gddiminfo(gridid, "Bands")
```

Duplicate a Region or Period

HE5_GDdupregion

hid_t HE5_GDdupregion(hid_t *oldregionID*)

| | |
|--------------------|--|
| <i>oldregionID</i> | IN: Region or period ID returned by HE5_GDdefboxregion, HE5_GDdeftimeperiod, or HE5_GDdefvrtregion. |
| Purpose | Duplicates a region. |
| Return value | Returns new region or period ID if successful or FAIL (-1) otherwise. |
| Description | This routine copies the information stored in a current region or period to a new region or period and generates a new id. It is usefully when the user wishes to further subset a region (period) in multiple ways. |
| Example | In this example, we first subset a grid with <i>HE5_GDdefboxregion</i> , duplicate the region creating a new region ID, <i>regionID2</i> , and then perform two different vertical subsets of these (identical) geographic subset regions: |

```
regionID = HE5_GDdefboxregion(gridID, cornerlon, cornerlat);
regionID2 = HE5_GDdupregion(regionID);
regionID = HE5_GDdefvrtregion(gridID, regionID, "Pressure",
rangePres);
regionID2 = HE5_GDdefvrtregion(gridID, regionID2,
"Temperature", rangeTemp);
```

| | |
|---------|--|
| FORTRAN | integer he5_gddupreg(<i>oldregionid</i>) |
| | integer <i>oldregionid</i> |

The equivalent *FORTRAN* code for the example above is:

```
regionid = he5_gddefboxreg(gridid, cornerlon, cornerlat)
regionid2 = he5_gddupreg(regionid)
regionid = he5_gddefvrtreg(gridid, regionid, 'Pressure',
rangePres)

regionid2 = he5_gddefvrtreg(gridid, regionid2,
'Temperature', rangeTemp)
```

Read a Region of interest from a Field

HE5_GDextractregion

```
herr_t HE5_GDextractregion(hid_t gridID, hid_t regionID, const char *fieldname,  
                           void *buffer)
```

| | |
|------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>regionID</i> | IN: Region (period) ID returned by HE5_GDdefboxregion (HE5_GDdeftimeperiod) |
| <i>fieldname</i> | IN: Field to subset |
| <i>buffer</i> | OUT: Data Buffer |
| Purpose | Extracts (reads) from subsetted region. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine reads data into the data buffer from a subsetted region as defined by <i>HE5_GDdefboxregion</i> . |
| Example | In this example, we extract data from the <i>Temperature</i> field from the region defined in <i>HE5_GDdefboxregion</i> . We first allocate space for the data buffer. The size of the subsetted region for the field is given by the HE5_GDregioninfo routine. |

```
datbuf = (float *)calloc(size, sizeof(float));  
  
status = HE5_GDextractregion(GDid, regionID, "Temperature",  
                             datbuf);
```

FORTRAN integer function he5_gdextreg(*gridid*, *regionid*, *fieldname*, *datbuf*)
 integer *gridid*
 integer *regionid*
 character(*) *fieldname*
 <valid type> *buffer(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdextreg(gridid, regionid, "Temperature",  
                      datbuf)
```

Retrieve Information about Data Field in a Grid

HE5_GDfieldinfo

```
herr_t HE5_GDfieldinfo(hid_t gridID, const char *fieldname, int *rank, hsize_t  
                      dims[], H5T_class_t ntype[], char *dimlist, char *maxdimlist)
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Fieldname |
| <i>rank</i> | OUT: Pointer to rank of the field |
| <i>dims</i> | OUT: Array containing the dimension sizes of the field |
| <i>ntype</i> | OUT: Pointer to the numbertype of the field |
| <i>dimlist</i> | OUT: Dimension list |
| <i>maxdimlist</i> | OUT: Maximum dimensions allowed for field |
| Purpose | Retrieve information about a specific geolocation or data field in the grid. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. A typical reason for failure is the specified field does not exist. |
| Description | This routine retrieves information on a specific data field. |
| Example | In this example, we retrieve information about the <i>Spectra</i> data fields: |

```
status = HE5_GDfieldinfo(gridID, "Spectra", &rank, dims,  
                         &ntype, dimlist, maxdimlist);
```

The return parameters will have the following values:

rank=3, *ntype*=1, *dims*[3]={15,200,120} and
dimlist="Bands,YDim,XDim"

FORTRAN integer function he5_gdflinfo(*gridid*, *fieldname*, *rank*, *dims*, *ntype*,
 dimlist, *maxdimlist*)

 integer *gridid*
 character(*) *fieldname*
 integer(*) *rank*

```
integer*4      dims(*)
integer        ntype(*)
character(*)   dimlist
character(*)   maxdimlist
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdfldinfo(gridid, "Spectra", dims, rank, ntype,
dimlist, maxdimlist)
```

The return parameters will have the following values:

rank=3, *ntype*=1, *dims*[3]={120,200,15} and
dimlist="XDim,YDim,Bands"

Note that the dimensions array and the dimension list are in *FORTRAN* order.

Get External Data File Information

HE5_GDgetextdata

```
int HE5_GDgetextdata(hid_t gridID, char *fieldname, size_t namelength, char  
                      *filelist, off_t offset[], hsize_t size[])
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: External field name |
| <i>namelength</i> | OUT: Length of each name entry |
| <i>filelist</i> | OUT: List of file names |
| <i>offset</i> [] | OUT: Array of offsets (in byte) from the beginning of file to the location in file where the data starts |
| <i>size</i> [] | OUT: Array of sizes (in bytes) reserved in the file for the data |
| Purpose | Retrieves information about external data file(s) associated with the data set. |
| Return value | Returns number of external data files if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid ID or field name. |
| Example | In this example, we get information about the <i>ExtData</i> field: |

```
nfiles = HE5_GDgetextdata(gridID, "ExtData", namlen,  
                           filenames, offset, size);
```

```
FORTRAN integer function he5_gdgetxdat(gridid,fieldname,nlen,flist,offset,size)  
        integer      gridid  
        integer      nfiles  
        integer*4    nlen  
        integer*4    offset(*)  
        integer*4    size(*)  
        character(*) fieldname  
        character(*) flist
```

The equivalent *FORTRAN* code for the example above is:

```
nfiles = he5_gdgetxdat(gridid, "ExtData",nlen,  
flist,offset,size)
```

Get Fill Value for Specified Field

HE5_GDgetfillvalue

herr_t HE5_GDgetfillvalue(hid_t *gridID*, const char **fieldname*, void **fillvalue*)

| | |
|------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Fieldname |
| <i>fillvalue</i> | OUT: Space allocated to store the fill value |
| Purpose | Retrieves fill value for the specified field. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper Grid ID or number type or incorrect fill value. |
| Description | It is assumed the number type of the fill value is the same as the field. |
| Example | In this example, we get the fill value for the <i>Temperature</i> field: |

```
status = HE5_GDgetfillvalue(gridID, "Temperature",  
&tempfill);
```

FORTRAN integer function he5_gdgetfill(*gridid*,*fieldname*,*fillvalue*)
 integer *gridid*
 character*(*) *fieldname*
 <valid type> *fillvalue*(*)

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdgetfill(gridid, "Temperature", tempfill)
```

Get Row/Columns for Specified Longitude/Latitude Pairs

HE5_GDgetpixels

```
herr_t HE5_GDgetpixels(hid_t gridID, long nLonLat, double lonVal[], double latVal[], long pixRow[], long pixCol[])
```

| | | |
|----------------|--|--|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>nLonLat</i> | IN: | Number of longitude/latitude pairs |
| <i>lonVal</i> | IN: | Longitude values in degrees |
| <i>latVal</i> | IN: | Latitude values in degrees |
| <i>pixRow</i> | OUT: | Pixel Rows |
| <i>pixCol</i> | OUT: | Pixel Columns |
| Purpose | Returns the pixel rows and columns for specified longitude/latitude pairs. | |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. | |
| Description | This routine converts longitude/latitude pairs into (0 - based) pixel rows and columns. The origin is the upper left-hand corner of the grid. This routine is the pixel subsetting equivalent of <i>HE5_GDdefboxregion</i> . | |
| Example | To convert two pairs of longitude/latitude values to rows and columns, make the following call: | |

```
lonArr[0] = 134.2;  
latArr[0] = -20.8;  
lonArr[1] = 15.8;  
latArr[1] = 84.6;  
  
status = HE5_GDgetpixels(gridID, 2, lonArr, latArr, rowArr,  
colArr);
```

The row and column of the two pairs will be returned in the *rowArr* and *colArr* arrays.

```

FORTRAN    integer function he5_gdgetpix(gridid, nlonlat, lonval, latval, pixrow,
          pixcol)
            integer      gridid
            integer*4    nlonlat
            real*8      lonval(*)
            real*8      latval(*)
            integer*4    pixrow(*)
            integer*4    pixcol(*)

```

The equivalent *FORTRAN* code for the example above is:

```

lonarr(1) = 134.2
latarr(1) = -20.8
lonarr(2) = 15.8
latarr(2) = 84.6
nlonlat   = 2
status = he5_gdgetpix(gridid, nlonlat, lonarr, latarr,
                      rowarr, colarr)

```

Note that the row and columns values will be 1 - based.

Get Field Values for Specified Row/Columns

HE5_GDgetpixvalues

```
long HE5_GDgetpixvalues(hid_t gridID, long nPixels, long pixRow[], long  
                  pixCol[], const char *fieldname, void *buffer)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

nPixels IN: Number of pixels

pixRow IN: Pixel Rows

pixCol IN: Pixel Columns

fieldname IN: Field from which to extract data values

buffer OUT: Buffer for data values

Purpose Read field data values for specified pixels.

Return value Returns size of data buffer if successful or FAIL(-1) otherwise.

Description This routine reads data from a data field for the specified pixels. It is the pixel subsetting equivalent of *HE5_GDextractregion*. All entries along the non-geographic dimensions (ie, NOT *XDim* and *YDim*) are returned. If the buffer is set to NULL, no data is returned but the data buffer size can be determined from the function return value.

Example To read values from the *Spectra* field with dimensions, *Bands*, *YDim*, and *XDim*, make the following call:

```
double        *datbuf;  
  
bufsiz = HE5_GDgetpixvalues(gridID, 2, rowArr, colArr,  
                  "Spectra", NULL);  
  
/* bufsiz will be equal to 2 * NBANDS * 8 where NBANDS is  
the value for the Bands dimension */  
  
datbuf = (double *)calloc(bufsiz, sizeof(double));  
  
bufsiz = HE5_GDgetpixvalues(gridID, 2, rowArr, colArr,  
                  "Spectra", datbuf);
```

```

FORTRAN  integer*4 function he5_gdgetpixval(gridid, npixels, pixrow, pixcol,
fieldname, buffer)
          integer      gridid
          integer*4     npixels
          integer*4     bufsiz
          integer*4     pixrow(*)
          integer*4     pixcol(*)
          character*(*) fieldname
          <valid type>  buffer(*)

```

The equivalent *FORTRAN* code for the example above is:

```

real*8       datbuf(2,NBANDS)

npixels = 2

bufsiz = he5_gdgetpixval(gridid, npixels, rowarr, colarr,
  "Spectra", datbuf)

```

Return Information about a Grid Structure

HE5_GDgridinfo

```
herr_t HE5_GDgridinfo(hid_t gridID, long *xdimsize, long *ydimsize, double  
                      upleftpt[], double lowrightpt[])
```

| | |
|-------------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>xdimsize</i> | OUT: Number of columns in grid |
| <i>ydimsize</i> | OUT: Number of rows in grid |
| <i>upleftpt</i> | OUT: Location, in meters, of upper left corner |
| <i>lowrightpt</i> | OUT: Location, in meters, of lower right corner |
| Purpose | Returns position and size of grid |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise |
| Description | This routine returns the number of rows, columns and the location, in meters, of the upper left and lower right corners of the grid image. |
| Example | In this example, we retrieve information from a previously created grid with a call to <i>HE5_GDattach</i> : |

```
status = HE5_GDgridinfo(gridID, &xdimsize, &ydimsize,  
                        upleft, lowrgt);
```

FORTRAN

```
integer function he5_gdgridinfo(gridid, xdimsize, ydimsize, upleftpt,  
                                 lowrightpt)
```

| | |
|-----------|----------------------|
| integer | <i>gridid</i> |
| integer*4 | <i>xdimsize</i> |
| integer*4 | <i>ydimsize</i> |
| real*8 | <i>upleftpt(2)</i> |
| real*8 | <i>lowrightpt(2)</i> |

The equivalent FORTRAN code for the example above is:

```
status = he5_gdgridinfo(gridid, xdimsize, ydimsize, upleft,  
                        lowrgt)
```

Return Information about a Group Grid Attribute

HE5_GDgrpattrinfo

```
herr_t HE5_GDgrpattrinfo(hid_t gridID, const char *attrname, H5T_class_t *ntype, hsize_t *count)
```

| | |
|-----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of attribute elements |
| Purpose | Returns information about a swath group attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a swath group attribute. |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_GDgrpattrinfo(gridID, "ScalarFloat", &ntype,  
&count);
```

The *ntype* variable will have the value 1 and *count* will have the value 1.

FORTRAN
integer function he5_gdgattrinfo(gridid, attrname, ntype, count,
integer gridid
character(*) attrname
integer ntype
integer *4 count

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_gdgattrinfo(gridid, "ScalarFloat", ntype,  
count)
```

Retrieve Information about Grid Attributes

HE5_GDinqattrs

long HE5_GDinqattrs(hid_t *gridID*, char **attrnames*, long **strbufsize*)

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about attributes defined in grid. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each attribute name separated by commas. If <i>attrnames</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the attributes defined in a grid structure. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = HE5_GDinqattrs(gridID, NULL, strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = HE5_GDinqattrs(gridID, attrnames, strbufsize);
```

The variable, *attrlist*, will be set to:

"*attrOne,attr_2*".

FORTRAN integer*4 function he5_gdinqattrs(*gridid*,*attrnames*,*strbufsize*)
integer *gridid*
character(*) *attrnames*
integer*4 *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_gdinqattrs(gridid, attrnames, strbufsize)
```

Retrieve Information about Dimensions Defined in Grid

HE5_GDinqdims

int HE5_GDinqdims(hid_t *gridID*, char **dimnames*, hsize_t *dims*[])

| | |
|-----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>dimnames</i> | OUT: Dimension list (entries separated by commas) |
| <i>dims</i> | OUT: Array containing size of each dimension |
| Purpose | Retrieve information about dimensions defined in grid. |
| Return value | Number of dimension entries found if successful or FAIL(-1) otherwise. A typical reason for failure is an improper grid id. |
| Description | The dimension list is returned as a string with each dimension name separated by commas. Output parameters set to <i>NULL</i> will not be returned. |
| Example | To retrieve information about the dimensions, use the following statement: |

```
ndim = HE5_GDinqdims(gridID, dimnames, dims);
```

The parameter, *dimnames*, will have the value: "Xgrid,Ygrid,Bands"

with *dims*[3]={120,200,15}

| | |
|---------|---|
| FORTRAN | integer function he5_gdinqdims(<i>gridid</i> , <i>dimnames</i> , <i>dims</i>) integer <i>gridid</i> character(*) <i>dimnames</i> integer*4 <i>dims</i> (*) |
|---------|---|

The equivalent *FORTRAN* code for the example above is:

```
ndim = he5_gdinqdims(gridid, dimnames, dims)
```

Retrieve Information about Data Fields Defined in Grid

HE5_GDinqfields

```
int HE5_GDinqfields(hid_t gridID, char *fieldlist, int rank[], H5T_class_t  
                      ntype[])
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldlist</i> | OUT: Listing of data fields (entries separated by commas) |
| <i>rank</i> | OUT: Array containing the rank of each data field |
| <i>numbertype</i> | OUT: Array containing the numbertype of each data field |
| Purpose | Retrieve information about the data fields defined in grid. |
| Return value | Number of data fields found if successful or FAIL(-1) otherwise. A typical reason is an improper grid id. |
| Description | The field list is returned as a string with each data field separated by commas. The <i>rank</i> and <i>numbertype</i> arrays will have an entry for each field. Output parameters set to <i>NULL</i> will not be returned. |
| Example | To retrieve information about the data fields, use the following statement: |

```
nfld = HE5_GDinqfields(gridID, fieldlist, rank, numbertype);
```

The parameter, *fieldlist*, will have the value: "*Temperature,Spectra*"

with *rank[2]={2,3}*, *numbertype[2]={1,1}*

FORTRAN integer function he5_gdinqdfls(*gridid, fieldlist, rank, numbertype*)
integer *gridid*
character*(*) *fieldlist*
integer *rank(*)*
integer *numbertype(*)*

The equivalent *FORTRAN* code for the example above is:

```
nfld = he5_gdinqdfls(gridID, fieldlist, rank, numbertype);
```

The parameter, *fieldlist*, will have the value: "*Spectra,Temperature*"

with *rank[2]={3,2}*, *numbertype[2]={1,1}*

Retrieve Grid Structures Defined in HDF-EOS File

HE5_GDinqgrid

long HE5_GDinqgrid(const char * *filename*, char **gridlist*, long **strbufsize*)

| | |
|-------------------|--|
| <i>filename</i> | IN: HDF-EOS file name |
| <i>gridlist</i> | OUT: Grid list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of grid list |
| Purpose | Retrieves number and names of grids defined in HDF-EOS file. |
| Return value | Number of grids found or FAIL (-1) otherwise. |
| Description | The grid list is returned as a string with each grid name separated by commas. If <i>gridlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . If <i>strbufsize</i> is also set to NULL, the routine returns just the number of grids. Note that <i>strbufsize</i> does not count the null string terminator. |
| Example | In this example, we retrieve information about the grids defined in an HDF-EOS file, <i>Grid.he5</i> . We assume that there are two grids stored, <i>GridOne</i> and <i>Grid_2</i> : <code>ngrid = HE5_GDinqgrid("Grid.he5", NULL, strbufsize);</code> The parameter, <i>ngrid</i> , will have the value 2 and <i>strbufsize</i> will have value 16. <code>ngrid = HE5_GDinqgrid("Grid.he5", gridlist, strbufsize);</code> The variable, <i>gridlist</i> , will be set to: “ <i>GridOne,Grid_2</i> ”. |
| FORTRAN | integer*4 function he5_gdinqgrid(<i>filename</i> , <i>gridlist</i> , <i>strbufsize</i>) character*(*) <i>filename</i> character*(*) <i>gridlist</i> integer*4 <i>strbufsize</i> The equivalent FORTRAN code for the example above is: <code>ngrid = he5_gdinqgrid('Grid.he5', gridlist, strbufsize)</code> |

Retrieve Information Grid Group Attributes

HE5_GDinqgrpattrs

long HE5_GDinqgrpattrs(hid_t *gridID*, char **attrnames*, long **strbufsize*)

| | |
|-------------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about group attributes defined in grid. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each group attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |

Example In this example, we retrieve information about the group attributes defined for the “Data Fields” group. We assume that there are two attributes stored, *attrOne* and *attr_2*:

```
nattr = HE5_GDinqgrpattrs(gridID, NULL, &strbufsize);
```

The parameter, *nattr*, will have the value 2 and *strbufsize* will have value 14.

```
nattr = HE5_GDinqgrpattrs(gridID, attrnames, &strbufsize);
```

The variable, *attrlist*, will be set to:

“*attrOne,attr_2*”.

FORTRAN integer*4 function he5_gdinqgattr(ιgridid*,* *attrnames*, *strbufsize*)
integer *gridid*
character(*) *attrnames*
integer*4 *strbufsize*

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_gdinqgattr(ιgridid, attrnames, strbufsize)
```

Retrieve Information Grid Local Attributes

HE5_GDinqlocattrs

```
long HE5_GDinqlocattrs(hid_t gridID, const char *fieldname, char *attrnames,  
                        long *strbufsize)
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Fieldname to retrieve local attribute information |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about local attributes defined for a field. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each local attribute name separated by commas. If <i>attrnames</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |
| Example | In this example, we retrieve information about the local attributes defined for a field “DataField”.. We assume that there are two attributes stored, <i>attrOne</i> and <i>attr_2</i> : |
| | <pre>nattr = HE5_GDinqlocattrs(gridID, "DataField", NULL, &strbufsize);</pre> |
| | The parameter, <i>nattr</i> , will have the value 2 and <i>strbufsize</i> will have value 14. |
| | <pre>nattr = HE5_GDinqlocattrs(gridID, "DataField", attrnames, &strbufsize);</pre> |
| | The variable, <i>attrnames</i> , will be set to: "attrOne,attr_2". |
| FORTRAN | <pre>integer*4 function he5_gdinqlatrs(gridid ,fieldname, attrnames, strbufsize) integer gridid character(*) fieldname character(*) attrnames</pre> |

```
integer*4      strbufsize
```

The equivalent *FORTRAN* code for the example above is:

```
nattr = he5_gdinglattr(gid, "DataField", attrnames,  
strbufsize)
```

Perform Bilinear Interpolation on Grid Field

HE5_GDinterpolate

```
long HE5_GDinterpolate(hid_t gridID, long nValues, double lonVal[], double latVal[], const char *fieldname, double interpVal[])
```

| | |
|------------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>nValues</i> | IN: Number of interpolation points |
| <i>lonVal</i> | IN: Longitude of interpolation points |
| <i>latVal</i> | IN: Latitude of interpolation points |
| <i>fieldname</i> | OUT: Field from which to interpolate data values |
| <i>interpVal</i> | OUT: Buffer for interpolated data values |
| Purpose | Performs bilinear interpolation on a grid field. |
| Return value | Returns size in bytes of interpolated data values if successful or FAIL(-1) otherwise. |
| Description | This routine performs bilinear interpolation on a grid field. It assumes that the pixel data values are uniformly spaced which is strictly true only for an infinitesimally small region of the globe but is a good approximation for a sufficiently small region. The default position of the pixel value is pixel center, however if the pixel registration has been set to HDFE_CORNER (with the <i>HE5_GDdefpixreg</i> routine) then the value is located at one of the four corners (HE5_HDFE_GD_UL, _UR, _LL, _LR) specified by the <i>HE5_GDdeforigin</i> routine. All entries along the non-geographic dimensions (ie, NOT XDim and YDim) are interpolated and all interpolated values are returned as DOUBLE. The data buffer size can be determined by setting the <i>interpVal</i> parameter to NULL. The reference for the interpolation algorithm is <i>Numerical Recipes in C</i> (2 nd ed). (Note for the current version of this routine, the number type of the field to be interpolated is restricted to INT, LONG, FLOAT, DOUBLE.) |
| Example | To interpolate the <i>Spectra</i> field at two geographic data points: |

```
lonVal[0] = 134.2;  
latVal[0] = -20.8;  
lonVal[1] = 15.8;  
latVal[1] = 84.6;
```

```

double      *interVal;

bufsiz = HE5_GD interpolate(gridID, 2, lonVal, latVal,
" Spectra ", NULL);

/* bufsiz will be equal to 2 * NBANDS * 8 where NBANDS is
the value for the Bands dimension */

interpVal = (double *)calloc(bufsiz, sizeof(double));

bufsiz = HE5_GD interpolate(gridID, 2, lonVal, latVal,
" Spectra ", interpVal);

```

FORTRAN integer*4 function he5_gd interpolate(*gridid*, *ninterp*, *lonval*, *latval*,
fieldname, *interpval*)

integer *gridid*
 integer*4 *ninterp*
 real*8 *lonval(*)*
 real*8 *latval(*)*
 character*(*) *fieldname*
 real*8 *interpval(*)*

The equivalent *FORTRAN* code for the example above is:

```

real*8      interpval(NBANDS, 2)

ninterp = 2

bufsiz = he5_gd interpolate(gridid, ninterp, lonval, latval,
" Spectra ", interpval)

```

Return Information about a Local Grid Attribute

HE5_GDlocattrinfo

```
herr_t HE5_GDlocattrinfo(hid_t gridID, const char *fieldname, const char  
*attrname, H5T_class_t *ntype, hsize_t *count)
```

| | |
|-------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Field name |
| <i>attrname</i> | OUT: Attribute list |
| <i>numbertype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of attribute elements |
| Purpose | Returns information about a Data Field's local attribute(s) |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of a data field's local attribute(s). |
| Example | In this example, we return information about the <i>ScalarFloat</i> attribute. |

```
status = HE5_GDlocattrinfo(gridID, "DataField", attrname,  
&ntype, &count);
```

The *ntype* variable will have the value 1 and *count* will have the value 1.

FORTRAN

```
integer function he5_gdlattrinfo(gridid, fieldname, attrname, ntype,  
count)  
integer      gridid  
character(*) fieldname  
character(*) attrname  
integer      ntype  
integer *4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_gdlattrinfo(gridid, "DataField", attrname,  
ntype, count)
```

Return Number of specified Objects in a Grid

HE5_GDnentries

long HE5_GDnentries(hid_t *gridID*, int *entrycode*, long **strbufsize*)

| | | |
|-------------------|------|--|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>entrycode</i> | IN: | Entry code |
| <i>strbufsize</i> | OUT: | String buffer size |
| Purpose | | Returns number of entries and descriptive string buffer size for a specified entity. |
| Return value | | Number of entries if successful or FAIL(-1) otherwise. A typical reason for failure is an improper Grid ID or entry code. |
| Description | | This routine can be called before using the inquiry routines in order to determine the sizes of the output arrays and descriptive strings. The string length does not include the NULL terminator. |
| | | The entry codes are: HE5_HDFE_NENTDIM (0) - Dimensions HE5_HDFE_NENTDFLD (4) - Data Fields |
| Example | | In this example, we determine the number of data field entries and the size of the field list string. |

ndims = HE5_GDnentries(gridID, HE5_HDFE_NENTDFLD, &bufsize);

FORTRAN integer*4 function he5_gdnentries(*gridid*,*entrycode*, *bufsize*)
integer *gridid*
integer *entrycode*
integer*4 *bufsize*

The equivalent *FORTRAN* code for the example above is:

```
entrycode = 4  
ndims = he5_gdnentries(gridid, entrycode, bufsize)
```

Open HDF-EOS File

HE5_GDopen

hid_t HE5_GDopen(const char *filename, uintn access)

| | |
|-----------------|--|
| <i>filename</i> | IN: Complete path and filename for the file to be opened |
| <i>access</i> | IN: H5F_ACC_RDONLY, H5F_ACC_RDWR or H5F_ACC_TRUNC |
| Purpose | Opens or creates HDF file in order to create, read, or write a grid. |
| Return value | Returns the grid file ID handle(<i>fid</i>) if successful or FAIL(-1) otherwise. |
| Description | This routine creates a new file or opens an existing one, depending on the access parameter. |

Access codes:

H5F_ACC_RDONLY Open for read only. If file does not exist,
error

H5F_ACC_RDWR Open for read/write. If file does not exist,
create it

H5F_ACC_TRUNC If file exists, delete it, then open a new file
for read/write

Example In this example, we create a new grid file named, *Grid.he5*. It returns the file handle, *fid*.

```
fid = HE5_GDopen( "Grid.he5" , H5F_ACC_TRUNC );
```

FORTRAN integer function he5_gdopen(*filename*, *access*)
character(*) *filename*
integer *access*

The access codes should be defined as parameters:

parameter (HE5_HDFE_RDWR=0)

parameter (HE5_HDFE_RDONLY=1)

parameter (HE5_HDFE_TRUNC=2)

The equivalent *FORTRAN* code for the example above is:

```
fid = he5_gdopen( "Grid.he5" , HE5_HDFE_TRUNC )
```

Note to users of the SDP Toolkit: Please refer to the *Release 5B SDP Toolkit User Guide for the ECS Project (333-CD-510-001)*, Section 6.2.1.2 for information on how to obtain a file name (referred to as a "physical file handle") from within a PGE. See also Section 9 of this document for code examples.

Return Grid Origin Information

HE5_GDorigininfo

herr_t HE5_GDorigininfo(hid_t *gridID*, int **origincode*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

origincode IN: Origin code

Purpose Retrieve origin code.

Return value Origin code if successful or FAIL (-1) otherwise.

Description This routine retrieves the origin code.

Example In this example, we retrieve the origin code defined in *HE5_GDdeforigin*.

```
status = HE5_GDorigininfo(gridID, &origincode);
```

The return value, *origincode*, will be equal to 3

FORTRAN integer function he5_gdorginfo(*gridid*,*origincode*)

integer *gridid*

integer(*) *origincode*

The equivalent *FORTRAN* code for the above example is :

```
status = he5_gdorginfo(gridid, origincode)
```

Return Pixel Registration Information

HE5_GDpixreginfo

herr_t HE5_GDpixreginfo(hid_t *gridID*, int **pixregcode*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

pixregcode IN: Pixel registration code

Purpose Retrieve pixel registration code.

Return value Pixel registration code if successful or FAIL (-1) otherwise.

Description This routine retrieves the pixel registration code.

Example In this example, we retrieve the pixel registration code defined in *HE5_GDdefpixreg*.

```
status = HE5_GDpixreginfo(gridID, &pixregcode);
```

The return value, *pixregcode*, will be equal to 1

FORTRAN integer function he5_gdpreginfo(*gridid,pixregcode*)

```
integer      gridid
```

```
integer(*)   pixregcode
```

The equivalent *FORTRAN* code for the above example is :

```
status = he5_gdpreginfo(gridid, pixregcode)
```

Retrieve Grid Projection Information

HE5_GDprojinfo

```
herr_t HE5_GDprojinfo(hid_t gridID, int *projcode, int *zonecode, int  
*spherecode, double projparm[])
```

| | |
|-------------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>projcode</i> | OUT: GCTP projection code |
| <i>zonecode</i> | OUT: GCTP zone code used by UTM projection |
| <i>spherecode</i> | OUT: GCTP spheroid code |
| <i>projparm</i> | OUT: GCTP projection parameter array |
| Purpose | Retrieves projection information of grid |
| Return Value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise |
| Description | Retrieves the GCTP projection code, zone code, spheroid code and the projection parameters of the grid |
| Example | In this example, we are retrieving the projection information from a grid attached to with <i>HE5_GDattached</i> : |

```
status = HE5_GDprojinfo(gridID, &projcode, &zonecode,  
&spherecode, projparm);
```

FORTRAN integer function he5_gdprojinfo(*gridid*, *projcode*, *zonecode*, *spherecode*,
 projparm)
 integer(*) *gridid*
 integer(*) *projcode*
 integer(*) *zonecode*
 integer(*) *spherecode*
 real*8 *projparm*(*)

The equivalent FORTRAN code for the example above is:

```
status = he5_gdprojinfo(gridid, projcode, zonecode,  
spherecode, projparm)
```

Read Grid Attribute

HE5_GDreadattr

herr_t HE5_GDreadattr(hid_t *gridID*, const char **attrname*, void **datbuf*)

| | | |
|-----------------|------|---|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrname</i> | IN: | Attribute name |
| <i>datbuf</i> | OUT: | Buffer allocated to hold attribute values |
| Purpose | | Reads attribute from a grid. |
| Return value | | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type or incorrect attribute name. |
| Description | | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_GDreadattr(gridID, "ScalarFloat", &attr_val);
```

FORTRAN
integer function he5_gdrdattr(*gridid*, *attrname*,*datbuf*)
integer *gridid*
character*(*)*attrname*
<valid type> *attrval(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdrdattr(gridid, "ScalarFloat", attrval)
```

Read Data From a Grid Field

HE5_GDreadfield

```
herr_t HE5_GDreadfield(hid_t gridID, const char *fieldname, const hssize_t  
                      start[], const hsize_t stride[], const hsize_t edge[], void  
                      *buffer)
```

| | | |
|------------------|--|---|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: | Name of field to read |
| <i>start</i> | IN: | Array specifying the starting location within each dimension |
| <i>stride</i> | IN: | Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: | Array specifying the number of values to write along each dimension |
| <i>buffer</i> | IN: | Buffer to store the data read from the field |
| Purpose | Reads data from a grid field. | |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are improper Grid ID or unknown fieldname. | |
| Description | The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the grid field (input) dimensions. The output data in <i>buffer</i> is written to contiguously. The default values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are <i>(dim - start) / stride</i> where <i>dim</i> refers to the size of the dimension. | |
| Example | In this example, we read data from the 10th row (0-based) of the <i>Temperature</i> field. | |
| | <pre>float row[120]; hssize_t start[2]={10,1}; hsize_t edge[2]={1,120}; status = HE5_GDreadfield(gridID, "Temperature", start, NULL, edge, row);</pre> | |

FORTRAN integer function

```
he5_gdrdfld(gridid,fieldname,start,stride,edge,buffer)
integer      gridid
character*(*) fieldname
integer*4     start(*)
integer*4     stride(*)
integer*4     edge(*)
<valid type> buffer(*)
```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```
real*4 row(2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1
status = he5_gdrdfld(gridid, "Temperature", start, stride,
edge, row)
```

Read Group Grid Attribute

HE5_GDreadgrpattr

herr_t HE5_GDreadgrpattr(hid_t *gridID*, const char **attrname*, void **datbuf*)

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

attrname IN: Attribute name

datbuf OUT: Buffer allocated to hold attribute values

Purpose Reads attribute from a grid.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid id or number type or incorrect attribute name.

Description The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types.

Example In this example, we read a floating point attribute with the name "ScalarFloat":

```
status = HE5_GDreadgrpattr(gridID, "ScalarFloat",  
&attr_val);
```

FORTRAN integer function he5_gdrdgattr(*gridid*,*attrname*,*datbuf*)

integer *gridid*

character(*) *attrname*

<valid type> *attrval(*)*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdrdgattr(gridid, "ScalarFloat", attrval)
```

Read Local Grid Attribute

HE5_GDreadlocattr

```
herr_t HE5_GDreadlocattr(hid_t gridID, const char *fieldname, const char  
*attrname, void *datbuf)
```

| | |
|------------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: Field name |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | OUT: Buffer allocated to hold attribute values |
| Purpose | Reads attribute from a grid. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper Grid ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read a floating point attribute with the name "ScalarFloat": |

```
status = HE5_GDreadlocattr(gridID, "DataField",  
"ScalarFloat", &attr_val);
```

FORTRAN

```
integer function he5_gdrdlatr(gridid, fieldname, attrname, datbuf)  
integer gridid  
character(*) fieldname  
character(*) attrname  
<valid type> attrval(*)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdrdlatr(gridid, "DataField", "ScalarFloat",  
attrval)
```

Return Information about a Region

HE5_GDregioninfo

```
herr_t HE5_GDregioninfo(hid_t gridID, hid_t regionID, const char * fieldname,  
                         H5T_class_t *ntype, int *rank, hsize_t dims[], long  
                         *size, double upleftpt[], double lowrightpt[])
```

| | |
|-------------------|--|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>regionID</i> | IN: Region (period) ID returned by HE5_GDdefboxregion (HE5_GDdeftimeperiod) |
| <i>fieldname</i> | IN: Field to subset |
| <i>ntype</i> | OUT: Number type of field |
| <i>rank</i> | OUT: Rank of field |
| <i>dims</i> | OUT: Dimensions of subset region |
| <i>size</i> | OUT: Size in bytes of subset region |
| <i>upleftpt</i> | OUT: Upper left point of subset region |
| <i>lowrightpt</i> | OUT: Lower right point of subset region |
| Purpose | Retrieves information about the subsetted region. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns information about a subsetted region for a particular field. It is useful when allocating space for a data buffer for the region. Because of differences in number type and geolocation mapping, a given region will give different values for the dimensions and size for various fields. The <i>upleftpt</i> and <i>lowrightpt</i> arrays can be used when creating a new grid from the subsetted region. |
| Example | In this example, we retrieve information about the region defined in <i>HE5_GDdefboxregion</i> for the <i>Temperature</i> field. We use this to allocate space for data in the subsetted region. |

```
status = HE5_GDregioninfo(GDid, regionID, "Temperature",  
&ntype, &rank, dims, &size, upleft, lowright);
```

```
FORTRAN    integer function he5_gdreginfo(gridid, regionid, fieldname, ntype, rank,  
           dims, size, upleftpt, lowrightpt)  
           integer      gridid  
           integer      gridid  
           character(*) fieldname  
           integer      ntype  
           integer      rank  
           integer*4    dims(*)  
           integer*4    size  
           real*8      upleftpt(2)  
           real*8      lowrightpt(2)
```

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdreginfo(gridid, regid, "Spectra", ntype,  
                      rank, dims, size, upleftpt, lowrightpt)
```

Set External Data File(s)

HE5_GDsetextdata

```
herr_t HE5_GDsetextdata(hid_t gridID, const char *filelist, off_t offset[], hsize_t  
                      size[])
```

| | |
|-----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>filelist</i> | IN: List of external file names |
| <i>offset[]</i> | IN: Array of offsets (in byte) from the beginning of file to the location in file where the data starts |
| <i>size[]</i> | IN: Array of sizes (in bytes) reserved in the file for the data |
| Purpose | Sets the external data file(s) associated with the data set. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid ID. |

Example In this example, we set the *ExtData* field:

```
status = HE5_GDsetextdata(gridID, "ext-1.dat,ext-2.dat,ext-  
3.dat", offset, size);
```

FORTRAN integer function he5_gdsetxdat(*gridid,fllist,offset, size*)
 integer *gridid*
 integer *status*
 integer*4 *offset(*)*
 integer*4 *size(*)*
 character*(*) *fllist*

The equivalent *FORTRAN* code for the example above is:

```
status = he5_gdsetxdat(gridid,flist,offset,size)
```

Set Fill Value for a Specified Field

HE5_GDsetfillvalue

```
herr_t HE5_GDsetfillvalue(hid_t gridID, const char *fieldname, hid_t ntype, void  
                           *fillvalue)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

fieldname IN: Fieldname

ntype IN: Data type of fill value

fillvalue IN: Pointer to the fill value to be used

Purpose Sets fill value for the specified field.

NOTE: **THIS FUNCTION MUST BE CALLED BEFORE THE FUNCTION CALL TO DEFINE THE FIELD IT IS TO BE APPLIED**

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type.

Description The fill value is placed in all elements of the field which have not been explicitly defined.

Example In this example, we set a fill value for the *Temperature* field:

```
tempfill = -999.0;  
  
status = HE5_GDsetfillvalue(gridID, "Temperature", ntype,  
&tempfill);
```

FORTRAN integer function

```
he5_gdsetfill(gridid,fieldname,ntype,fillvalue)  
integer*4      gridid  
character(*)   fieldname  
integer        ntype  
<valid type>  fillvalue(*)
```

The equivalent *FORTRAN* code for the example above is:

```
fillvalue = -999.0  
  
status = he5_gdsetfill(gridid, "Temperature", ntype,  
fillvalue)
```

Write/Update Grid Attribute

HE5_GDwriteattr

```
herr_t HE5_GDwriteattr(hid_t gridID, const char *attrname, hid_t ntype, hsize_t  
          count[], void *datbuf)
```

| | |
|-----------------|---|
| <i>gridID</i> | IN: Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | IN: Number type of attribute |
| <i>count</i> | IN: Number of values to store in attribute |
| <i>datbuf</i> | IN: Attribute values |
| Purpose | Writes/Updates attribute in a grid. |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. Typical reasons for failure are an improper grid id or number type. |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. |
| Example | In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14: <pre>attr_val = 3.14; count[0] = 1; status=HE5_GDwriteattr(gridid,"ScalarFloat",H5T_NATIVE_FLOAT , count, &attr_val);</pre> We can update this value by simply calling the routine again with the new value: <pre>attr_val = 3.14159; status=HE5_GDwriteattr(gridid,"ScalarFloat",H5T_NATIVE_FLOAT , count, &attr_val);</pre> |

```
FORTRAN    integer function he5_gdwrattr(gridid, attrname,  
           &          ntype, count, datbuf)  
           integer      gridid  
           character*(*) attrname  
           integer      ntype  
           integer*4     count(*)  
           <valid type> attrval(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)  
  
attrval = 3.14  
  
count(1)= 1  
  
status=he5_gdwrattr(gridid,"ScalarFloat",HE5_HDFE_NATIVE_FLOAT,  
                     count, attrval)
```

Write Data to a Grid Field

HE5_GDwritefield

```
herr_t HE5_GDwritefield(hid_t gridID, const char fieldname, const hsize_t  
                        start[], const hsize_t stride[], const hsize_t edge[], void data)
```

| | | |
|------------------|--|--|
| <i>gridID</i> | IN: | Grid ID returned by HE5_GDcreate or HE5_GDattach |
| <i>fieldname</i> | IN: | Name of field to write |
| <i>start</i> | IN: | Array specifying the starting location within each dimension (0-based) |
| <i>stride</i> | IN: | Array specifying the number of values to skip along each dimension |
| <i>edge</i> | IN: | Array specifying the number of values to write along each dimension |
| <i>data</i> | IN: | Values to be written to the field |
| Purpose | Writes data to a grid field. | |
| Return value | Returns SUCCEED(0) if successful or FAIL(-1) otherwise. | |
| Description | The values within <i>start</i> , <i>stride</i> , and <i>edge</i> arrays refer to the grid field (output) dimensions. The input data in the <i>data</i> buffer is read from contiguously. The default values for <i>start</i> and <i>stride</i> are 0 and 1 respectively and are used if these parameters are set to <i>NULL</i> . The default values for <i>edge</i> are <i>(dim - start) / stride</i> where <i>dim</i> refers to the size of the dimension. Note that the data buffer for a compressed field must be the size of the entire field as incremental writes are not supported by the underlying HDF routines. If this is not possible due to, for example, memory limitations, then the user should consider tiling. See <i>HE5_GDdefile</i> for further information. | |

Example In this example, we write data to the *Temperature* field.

```
float temperature [200][120];  
  
/* Define elements of temperature array */  
  
status = HE5_GDwritefield(gridID, "Temperature", NULL, NULL,  
                           NULL, temperature);
```

We now update Row 10 (0 - based) in this field:

```
float newrow[2000];  
  
hssize_t start[2]={0,10}; hsize_t edge[2]={2000,1};
```

```

/* Define elements of newrow array */

status = HE5_GDwritefield(gridID, "Temperature", start,NULL,
                           edge, newrow);

FORTRAN integer function he5_gdwrfld(gridid,fieldname,start,stride,edge,data)
integer      gridid
character(*) fieldname
integer*4     start(*)
integer*4     stride(*)
integer*4     edge(*)
<valid type> data(*)

```

The *start*, *stride*, and *edge* arrays must be defined explicitly, with the *start* array being 0-based.

The equivalent *FORTRAN* code for the example above is:

```

real*4 temperature(2000,1000)
integer*4 start(2), stride(2), edge(2)
start(1) = 0
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1000
status = he5_gdwrfld(gridid, "Temperature", start, stride,
                      edge, temperature)

```

We now update Row 10 (0 - based) in this field:

```

real*4 newrow(2000)
integer*4 start(2), stride(2), edge(2)
start(1) = 10
start(2) = 0
stride(1) = 1
stride(2) = 1
edge(1) = 2000
edge(2) = 1
status = he5_gdwrfld(gridid, "Temperature", start, stride,
                      edge, newrow)

```

Write Field Metadata for an Existing Field not Defined with the Grid API

HE5_GDwritefieldmeta

```
herr_t HE5_GDwritefieldmeta(hid_t gridID, const char *fieldname, char *dimlist,  
                           int ntype)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach
fieldname IN: Name of field that metadata information is to be written
dimlist IN: Dimension list of field
ntype IN: Number type of data in field
Purpose Writes field metadata for an existing grid field not defined with the Grid API

Return Value Returns SUCCEED(0) if successful or FAIL(-1) otherwise

Description This routine writes the field metadata for a grid field not defined by the Grid API

Example

```
status = HE5_GDwritefieldmeta(gridID, "ExternField",  
                               "Ydim,Xdim", HE5_HDFE_NATIVE_FLOAT);
```

FORTRAN integer function he5_gdwrmeta(*gridid*, *fieldname*, *dimlist*, *ntype*)
 integer *gridid*
 character*(*) *fieldname*
 character*(*) *dimlist*
 integer *ntype*

The equivalent *FORTRAN* code for the example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)  
  
status = he5_gdwrmeta(gridid, "ExternField"1, "Xdim,Ydim",  
                      HE5_HDFE_NATIVE_FLOAT)
```

Write/Update Group Grid Attribute

HE5_GDwritegrpattr

```
herr_t HE5_GDwritegrpattr(hid_t gridID, const char *attrname, hid_t ntype,
                           hsize_t count[], void *datbuf)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

attrname IN: Attribute name

ntype IN: Data type of attribute

count IN: Number of values to store in attribute

datbuf IN: Attribute values

Purpose Writes/Updates group attribute in a grid.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid id or number type.

Description If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the “Data Fields” group in the swath file.

Example In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_GDwritegrpattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_GDwritegrpattr(gridid, "ScalarFloat",  
H5T_NATIVE_FLOAT, count, &attr_val);
```

```
FORTRAN    integer function he5_gdwrgattr(gridid, attrname, ntype, count, datbuf)
            integer      gridid
            character*(* ) attrname
            integer      ntype
            integer*4     count(*)
            <valid type>  attrval(*)
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)
attrval = 3.14
count(1) = 1
status = he5_gdwrgattr(gridid, "ScalarFloat",
HE5_HDFE_NATIVE_FLOAT,count, attrval)
```

Write/Update Local Grid Attribute

HE5_GDwritelocattr

```
herr_t HE5_GDwritelocattr(hid_t gridID, const char *fieldname, const char  
*attrname, hid_t ntype, hsize_t count[], void *dbuf)
```

gridID IN: Grid ID returned by HE5_GDcreate or HE5_GDattach

fieldname IN: Field name

attrname IN: Attribute name

ntype IN: Data type of attribute

count IN: Number of values to store in attribute

dbuf IN: Attribute values

Purpose Writes/Updates group attribute in a grid.

Return value Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper grid id or number type.

Description If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to a particular “Data Field” in the grid file.

Example In this example, we write a floating point number with the name "ScalarFloat" and the value 3.14:

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_GDwritelocattr(gridid, "DataField",  
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_GDwritelocattr(gridid, "DataField",  
"ScalarFloat", H5T_NATIVE_FLOAT, count, &attr_val);
```

```
FORTRAN    integer function he5_gdwrlattr(gridid, fieldname, attrname, ntype, count,  
          &           datbuf)  
          integer      gridid  
          character*(* ) fieldname  
          character*(* ) attrname  
          integer*4     count(* )  
          <valid type>  attrval(* )
```

The equivalent *FORTRAN* code for the first example above is:

```
parameter (HE5_HDFE_NATIVE_FLOAT=1)  
  
attrval = 3.14  
  
count(1) = 1  
  
status = he5_gdwrlattr(gridid, "DataField", "ScalarFloat",  
HE5_HDFE_NATIVE_FLOAT, count, attrval)
```

2.1.4 HDF-EOS Utility Routines

This section contains an alphabetical list of the utility functions. The functions are alphabetized on their C-language names.

Convert Among Angular Units

HE5_EHconvAng

double HE5_EHconvAng(double *inAngle*, int *code*)

inAngle IN: Input angle

code IN: Conversion code

Purpose Convert among various angular units.

Return value Returns angle in desired units if successful or FAIL (-1) otherwise.

Description This routine converts angles between three units, decimal degrees, radians, and packed degrees-minutes-seconds. In the later unit, an angle is expressed as a integral number of degrees and minutes and a float point value of seconds packed as a single double number as follows: DDDMMSSS.SS. The six conversion codes are: HE5_HDFE_RAD_DEG (0), HE5_HDFE_DEG_RAD (1), HE5_HDFE_DMS_DEG (2), HE5_HDFE_DEG_DMS (3), HE5_HDFE_RAD_DMS (0), and HE5_HDFE_DMS_RAD (1), where the first three letter code (RAD - radians, DEG - decimal degrees, DMS - packed degrees-minutes-seconds) corresponds to the input angle and the second to the desired output angular unit.

Example To convert 27.5 degrees to packed format:

```
inAng = 27.5;  
outAng = HE5_EHconvAng(inAng, HDFE_DEG_DMS);  
“outAng” will contain the value: 27030000.00.
```

FORTRAN real*8 function he5_ehconvang(*inangle*,*code*)

real*8 *inangle*

integer *code*

The equivalent *FORTRAN* code for the example above is:

```
inangle = 27.5  
code = 3  
outangle = he5_ehconvang(inangle, code)
```

Get HDF-EOS Version String

HE5_EHgetversion

herr_t HE5_EHgetversion(hid_t *fid*, char **version*)

fid IN: File ID returned by *HE5_SWopen*, *HE5_GDopen*, or *HE5_PTopen*.

version OUT: HDF-EOS version string

Purpose Get HDF-EOS version string.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This routine returns the HDF-EOS version string of an HDF-EOS file. This designates the version of HDF-EOS that was used to create the file. This string is of the form: “HDFEOS_V*maj.min*” where *maj* is the major version and *min* is the minor version.

Example To get the HDF-EOS version (assumed to be 5.1) used to create the HDF-EOS file: “Swath.he5”:

```
char version[16];  
  
fid = HE5_SWopen("Swath.he5", H5F_ACC_RDONLY);  
  
status = HE5_EHgetversion(fid, version);  
  
“version” will contain the string: “HDFEOS_5.1”.
```

FORTRAN integer function he5_ehgetver(*fid*,*version*)

```
integer      fid  
character*(*) version  
integer      HE5_HDFE_RDONLY  
parameter    (HE5_HDFE_RDONLY=1)
```

The equivalent *FORTRAN* code for the example above is:

```
character*16 version  
  
fid = he5_swopen("Swath.he5",HE5_HDFE_RDONLY)  
  
status = he5_ehgetver(fid, version)
```

Return Information about Global File Attribute

HE5_EHglbattrinfo

```
herr_t HE5_EHglbattrinfo(hid_t fileID, const char *attrname, H5T_class_t *ntype,  
                           hsize_t *count)
```

| | |
|-----------------|---|
| <i>fileID</i> | IN: HDF-EOS file ID returned by HE5_SWopen/HE5_GDopen/HE5_PTopen |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | OUT: Data type class ID of attribute |
| <i>count</i> | OUT: Number of attribute elements |
| Purpose | Returns information about Global File attribute |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. |
| Description | This routine returns number type and number of elements (count) of Global File attribute. |
| Example | In this example, we return information about the <i>FloatAttr</i> attribute. |

```
status = HE5_EHglbattrinfo(fileID, "FloatAttr", &nt,  
&count);
```

The *nt* variable will have the value 1 and *count* will have the value 1.

FORTRAN

```
integer function he5_ehglatinf(fileid, attrname, ntype, count,)  
integer      fileid  
character(*) attrname  
integer      ntype  
integer *4    count
```

The equivalent *FORTRAN* code for the first example above is:

```
status = he5_ehglatinf(fileid, "FloatAttr", nt, count)
```

Get HDF-EOS File IDs

HE5_EHidinfo

herr_t HE5_EHidinfo(hid_t *fid*, hid_t **HDFfid*, hid_t **gid*)

fid IN: File ID returned by *HE5_SWopen*, *HE5_GDopen*, or *HE5_PTopen*.

HDFfid OUT: HDF-EOS file ID (returned by *HE5_EHopen*)

gid OUT: "HDFEOS" group ID

Purpose Get HDF-EOS file IDs.

Return value Returns SUCCEED(0) if successful or FAIL(-1) otherwise.

Description This is a wrapper around *HE5_EHchkfid* () and it returns the HDF file IDs to the HDF-EOS file ID returned by *HE5_SWopen*, *HE5_GDopen*, or *HE5_PTopen*. These ids can then be used to create or access HDF5 structures such as groups, attributes, datasets within an HDF-EOS file.

Retrieve Information about Global File Attributes

HE5_EHinqglbatts

long HE5_EHinqglbatts(hid_t *fileID*, char **attrnames*, long **strbufsize*)

| | |
|-------------------|--|
| <i>fileID</i> | IN: HDF-EOS file ID returned by HE5_SWopen/HE5_GDopen/HE5_PTopen |
| <i>attrnames</i> | OUT: Attribute list (entries separated by commas) |
| <i>strbufsize</i> | OUT: String length of attribute list |
| Purpose | Retrieve information about Global attributes defined in file. |
| Return value | Number of attributes found if successful or FAIL (-1) otherwise. |
| Description | The attribute list is returned as a string with each group attribute name separated by commas. If <i>attrlist</i> is set to NULL, then the routine will return just the string buffer size, <i>strbufsize</i> . This variable does not count the null string terminator. |
| Example | In this example, we retrieve information about the Global attributes defined for the “swath.he5” file (with the file ID <i>fileID</i>). We assume that there are two attributes stored, <i>GlobAttr_1</i> and <i>GlobAttr_2</i> : <code>nattr = HE5_EHinqglbatts(fileID, NULL, &strbufsize);</code> The parameter, <i>nattr</i> , will have the value 2 and <i>strbufsize</i> will have value 21. <code>nattr = HE5_EHinqglbatts(fileID, attrnames, &strbufsize);</code> The variable, <i>attrnames</i> , will be set to: "GlobAttr_1,GlobAttr_2". |
| FORTRAN | integer*4 function he5_ehqglatts(<i>fileid</i> , <i>attrnames</i> , <i>strbufsize</i>) integer <i>fileid</i> character*(*) <i>attrnames</i> integer*4 <i>strbufsize</i> integer*4 <i>nattr</i> The equivalent FORTRAN code for the example above is: <code>nattr = he5_ehqglatts(fileid, attrnames, strbufsize)</code> |

Read Global File Attribute

HE5_EHreadglbattr

herr_t HE5_EHreadglbattr(hid_t *fileID*, const char **attrname*, void **datbuf*)

| | |
|-----------------|---|
| <i>fileID</i> | IN: HDF-EOS file ID returned by HE5_SWopen/HE5_GDopen/HE5_PTopen |
| <i>attrname</i> | IN: Attribute name |
| <i>datbuf</i> | OUT: Buffer allocated to hold attribute values |
| Purpose | Reads global attribute from a file. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper file ID or number type or incorrect attribute name. |
| Description | The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. |
| Example | In this example, we read a floating point attribute with the name "FloatAttr": |

```
status = HE5_EHreadglbattr(fileID, "FloatAttr", &data);
```

FORTRAN integer function he5_ehrdglatt(*fileid*,*attrname*,*datbuf*)
integer *fileid*
character*(*), intent(in) :: *attrname*
<valid type> *datbuf*(*), intent(out)

The equivalent *FORTRAN* code for the example above is:

```
status = he5_ehrdglatt(fileid, "FloatAttr", datbuf)
```

Write/Update Global File Attribute

HE5_EHwriteglbattr

```
herr_t HE5_EHwriteglbattr(hid_t fileID, const char *attrname, hid_t ntype,  
                           hsize_t count[], void *datbuf)
```

| | |
|-----------------|--|
| <i>fileID</i> | IN: HDF-EOS file ID returned by HE5_SWopen/HE5_GDopen/HE5_PTopen |
| <i>attrname</i> | IN: Attribute name |
| <i>ntype</i> | IN: Data type of attribute |
| <i>count</i> | IN: Number of values to store in attribute |
| <i>datbuf</i> | IN: Attribute values |
| Purpose | Writes/Updates Global attribute in HDF-EOS file. |
| Return value | Returns SUCCEED (0) if successful or FAIL (-1) otherwise. Typical reasons for failure are an improper file ID or number type. |
| Description | If the attribute does not exist, it is created. If it does exist, then the value(s) is (are) updated. The attribute is passed by reference rather than value in order that a single routine suffice for all numerical types. Because of this a literal numerical expression should not be used in the call. The attribute is linked to the “ADDITIONAL/FILE ATTRIBUTES” group in the HDF-EOS file. |
| Example | In this example, we write a single precision (32 bit) floating point number with the name "FloatAttr" and the value 3.14: |

```
count[0] = 1;  
  
attr_val = 3.14;  
  
status = HE5_EHwriteglbattr(fileid, "FloatAttr",  
                           H5T_NATIVE_FLOAT, count, &attr_val);
```

We can update this value by simply calling the routine again with the new value:

```
attr_val = 3.14159;  
  
status = HE5_EHwriteglbattr(fileid, "FloatAttr",  
                           H5T_NATIVE_FLOAT, count, &attr_val);
```

```
FORTRAN    integer function he5_ehwrglbattr(fid,attrname,ntype,count,buffer)
            integer      fid,status,ntype
            character *(*) attrname
            integer*4     count
            <valid type>   buffer(*)
            integer       HE5_HDFE_NATIVE_FLOAT
            parameter     (HE5_HDFE_NATIVE_FLOAT=1)
```

The equivalent *FORTRAN* code for the example above is:

```
count = 1
status = he5_ptwrglbattr(fid, "FloatAttr",
                         HE5_NATIVE_FLOAT, count, buffer)
```

Abbreviations and Acronyms

| | |
|--------|--|
| AI&T | Algorithm Integration & Test |
| AIRS | Atmospheric Infrared Sounder |
| API | application program interface |
| ASTER | Advanced Spaceborne Thermal Emission and Reflection Radiometer |
| CCSDS | Consultative Committee on Space Data Systems |
| CDRL | Contract Data Requirements List |
| CDS | CCSDS day segmented time code |
| CERES | Clouds and Earth Radiant Energy System |
| CM | configuration management |
| COTS | commercial off-the-shelf software |
| CUC | constant and unit conversions |
| CUC | CCSDS unsegmented time code |
| DAAC | distributed active archive center |
| DBMS | database management system |
| DCE | distributed computing environment |
| DCW | Digital Chart of the World |
| DEM | digital elevation model |
| DTM | digital terrain model |
| ECR | Earth centered rotating |
| ECS | EOSDIS Core System |
| EDC | Earth Resources Observation Systems (EROS) Data Center |
| EDHS | ECS Data Handling System |
| EDOS | EOSDIS Data and Operations System |
| EOS | Earth Observing System |
| EOSAM | EOS AM Project (morning spacecraft series) |
| EOSDIS | Earth Observing System Data and Information System |
| EOSPM | EOS PM Project (afternoon spacecraft series) |

| | |
|--------|---|
| ESDIS | Earth Science Data and Information System (GSFC Code 505) |
| FDF | flight dynamics facility |
| FOV | field of view |
| ftp | file transfer protocol |
| GCT | geo-coordinate transformation |
| GCTP | general cartographic transformation package |
| GD | grid |
| GPS | Global Positioning System |
| GSFC | Goddard Space Flight Center |
| HDF | hierarchical data format |
| HITC | Hughes Information Technology Corporation |
| http | hypertext transport protocol |
| I&T | integration & test |
| ICD | interface control document |
| IDL | interactive data language |
| IP | Internet protocol |
| IWG | Investigator Working Group |
| JPL | Jet Propulsion Laboratory |
| LaRC | Langley Research Center |
| LIS | Lightening Imaging Sensor |
| M&O | maintenance and operations |
| MCF | metadata configuration file |
| MET | metadata |
| MODIS | Moderate-Resolution Imaging Spectroradiometer |
| MSFC | Marshall Space Flight Center |
| NASA | National Aeronautics and Space Administration |
| NCSA | National Center for Supercomputer Applications |
| netCDF | network common data format |
| NGDC | National Geophysical Data Center |
| NMC | National Meteorological Center (NOAA) |

| | |
|-------|---|
| ODL | object description language |
| PC | process control |
| PCF | process control file |
| PDPS | planning & data production system |
| PGE | product generation executive (formerly product generation executable) |
| POSIX | Portable Operating System Interface for Computer Environments |
| PT | point |
| QA | quality assurance |
| RDBMS | relational data base management system |
| RPC | remote procedure call |
| RRDB | recommended requirements database |
| SCF | Science Computing Facility |
| SDP | science data production |
| SDPF | science data processing facility |
| SGI | Silicon Graphics Incorporated |
| SMF | status message file |
| SMP | Symmetric Multi-Processing |
| SOM | Space Oblique Mercator |
| SPSO | Science Processing Support Office |
| SSM/I | Special Sensor for Microwave/Imaging |
| SW | swath |
| TAI | International Atomic Time |
| TBD | to be determined |
| TDRSS | Tracking and Data Relay Satellite System |
| TRMM | Tropical Rainfall Measuring Mission (joint US – Japan) |
| UARS | Upper Atmosphere Research Satellite |
| UCAR | University Corporation for Atmospheric Research |
| URL | universal reference locator |
| USNO | United States Naval Observatory |
| UT | universal time |

| | |
|------|-----------------------------------|
| UTC | Coordinated Universal Time |
| UTCF | universal time correlation factor |
| UTM | universal transverse mercator |
| VPF | vector product format |
| WWW | World Wide Web |